

LabVIEW for Control Analysis and Design

Todd Murphey

DRAFT as of October 7, 2008at 5:25 PM

Contents

Preface	7
1 Introduction	9
1.1 Introduction	9
1.1.1 Background and Purpose	9
1.1.2 Philosophical relationship between LabVIEW and other programming languages.	10
1.1.3 Organization	10
2 Basics of Graphical Programming	11
2.1 Graphical Programming as Data Flow	11
3 Simulation Module	13
3.1 Simulation in LabVIEW	13
3.1.1 Creating Sub-Systems in your Simulation	16
3.2 MathScript—a command line utility	16
3.3 Graphs in LabVIEW Simulation Module	18
3.3.1 Some Notes on Graphing in LabVIEW	20
3.4 LabVIEW Saving and Reading	21
4 Analysis	25
4.1 Frequency Domain Analysis	25
4.1.1 Root Locus	25
4.1.2 Bode	27
4.1.3 Nyquist	27
4.2 State Space Analysis	27
4.2.1 Stability	27
4.2.2 LQR	27
4.3 Digital Control	27

5	LabVIEW 8.2 Installation	29
5.1	Image Installation	29
5.2	Installation	29
5.2.1	Automatic Installation	30
5.2.2	Manual Installation	30
5.3	Installing the FPGA	31
5.4	Configuring LabVIEW	31
5.5	Using the NI-7831 FPGA with the ECP Model 205a Torsional Plant	32
5.6	Using the <code>reading_encoders.vi</code> FPGA interface VI.	34
5.7	Notes	35
6	Torsional Plant System Identification	37
6.1	System Identification	37
6.1.1	Rotational Inertia	38
6.1.2	Spring and Damper Coefficients	38
6.1.3	Hardware Gain	38
6.2	VI Implementations	39
6.2.1	System ID Recorder	39
6.2.2	System ID Analyzer	40
6.2.3	Hardware Gain Recorder	41
6.2.4	Hardware Gain Analyzer	42
A	Lab #1: Introduction to Digital Simulation in LabVIEW	45
A.1	Tasks	45
A.1.1	Task #1–Step Response	46
A.1.2	Task #2–Actuator Saturation	46
A.1.3	Task #3–Time Delay	46
A.1.4	Task #4–”Best” K_p	46
A.1.5	Task #5–Dependence on ω	47
A.2	Things You May Want To Know	47
B	Lab #2: Digital Simulation of Torsional Disk Systems in LabVIEW	49
B.1	Pre-Lab Tasks	50
B.2	Tasks	50
B.2.1	Task #1–Mechanical Model	50
B.2.2	Task #2–Simulation	52
B.2.3	Task #3–Sensitivity to Feedback	52
B.3	Lab #2 Final Tasks	53

C Lab #3: System Identification of the Torsional Disk System	55
C.1 Pre-Lab Tasks	56
C.2 Tasks	56
C.2.1 Task #1–Using the Hardware	56
C.2.2 Task #2–Creating a System Identification Procedure	57
C.2.3 Task #3–System Identification Experiment	57
C.2.4 Task #4–System Identification Results	58
C.2.5 Task #5–Hardware Gain	58
C.2.6 Task #6–System Identification Analysis	58
C.3 Things You May Want To Know	59
D Lab #4: Open-Loop and Hardware-Based PID Control Design	61
D.1 Pre-Lab Tasks	62
D.2 Tasks	62
D.2.1 Task #1–Open-Loop Control	62
D.2.2 Task #2–P, PD, PID Control	62
D.2.3 Task #3–Simulation	63
D.3 Things You May Want To Know	63
E Lab #5: Root Locus Design	65
E.1 Pre-Lab Tasks	66
E.2 Tasks	66
E.2.1 Task #1–Making Sure Your Simulation Works	66
E.2.2 Task #2–Root Loci for Disk 1 and Disk 3 Outputs	66
E.2.3 Task #3–Lead/Lag Controller	67
E.2.4 Task #4–Root Loci for Uncertainties	67
E.2.5 Task #5–Design for and Difficulties with Disk 3	67
E.3 Useful Things to Know	67
F Lab #6: Frequency Domain Compensator Design	69
F.1 Pre-Lab Tasks	70
F.2 Tasks	70
F.2.1 Task #1–Lead Controller for Disk 1	70
F.2.2 Task #2–Controller Design for Disk 3	70
G Lab #7: State Space Compensator Design	73
G.1 Pre-Lab Tasks	74
G.2 Tasks	74
G.2.1 Task #1–State Space simulation	74
G.2.2 Task #2–Controller Design	74

G.2.3	Task #3–Estimator Design	75
G.2.4	Task #4–Hardware Test	75

Preface

NOTE: KEEP IN MIND YOU ARE READING A *draft* OF THIS TUTORIAL THAT IS BEING MODIFIED AS THE STUDENTS USE IT (CURRENTLY IN AUTUMN, 2006). ALTHOUGH I APOLOGIZE FOR ANY UNFORESEEN CONFUSION, THIS IS NOT INTENDED TO BE A FINISHED PRODUCT YET.—TDM

This document has three primary goals:

1. It should introduce students to the minimal amount of graphical programming required to competently program embedded controllers. Although this is presented in terms of LabVIEW syntax (partially because this effort is being supported by a combination of the National Instruments Foundation and the National Science Foundation), the basic structure of how graphical programming can be used to allow students to do the majority of programming themselves is not dependent on LabVIEW. Other graphical programming languages (such as MATLAB/Simulink) can be employed in a nearly identical manner.
2. It should provide control systems laboratories that are open-ended enough to take advantage of the fact that students can write their own code. Among other things, this means that students are in charge of nearly all the implementation.
3. It should lastly provide a series of laboratories that complement a more modern approach to understanding control theory. In particular, the reader should note that there are almost no formulae presented here—instead the students are generally required to derive the relevant transformations and control laws themselves. (This is possible because they are not spending all their time programming in a more traditional language such as C/C++.)

The direction taken in this tutorial/laboratories is by no means unique and merely reflects the belief of the author that traditional laboratories have been too limited in what the students can try. We have had substantial success with the open-ended structure of the labs presented here, as documented in [?].

The reader should note that just like LabVIEW is not intrinsically a part of this laboratory structure, neither is the Educational Control Products (ECP) Torsional Disk System.

We are currently in the process of extending the basic structure of these labs to other experimental devices.

Lastly, the author would like to thank National Instruments and the National Instruments Foundation for their support of the laboratory. Moreover, the author would like to thank the National Science Foundation for supporting the ongoing assessment of this laboratory.
Todd Murphey

Chapter 1

Introduction

1.1 Introduction

LabVIEW is a graphical programming language in principle capable of the same utility that programming in C or C++ can provide. Some capabilities of C++ are more difficult to obtain, but for the purposes of control systems—the focus of this short tutorial—LabVIEW is an exceptionally convenient programming language. This is for two primary reasons:

1. LabVIEW enables programming that mirrors that graphical analysis tools (such as block diagrams) that we use to analyze control systems;
2. LabVIEW seamlessly (well, at least ideally seamlessly) incorporates “hardware-in-the-loop” needs into code.

We will see both of these become evident over the course of this tutorial.

1.1.1 Background and Purpose

This tutorial is being written as an accompaniment to the new control laboratory course ECEN 4638 [?] at the University of Colorado, based on student comments from previous laboratories. Hence, this tutorial is not intended to be exhaustive—exhaustive tutorials have not been very beneficial to students because they tend to be overwhelming. Instead, this tutorial aims to very specifically introduce students to LabVIEW in such a way that they can use it *in specific application to control systems* and so that they may become comfortable both with LabVIEW in particular and graphical programming in general, both of which are becoming standards in industry. Moreover, this type of exposure is intended to encourage more student investigation (both in LabVIEW, controls, and life in general) rather than explicitly laying out the solutions to all anticipated problems. (This learning philosophy is along the lines of [?].)

1.1.2 Philosophical relationship between LabVIEW and other programming languages.

LabVIEW is to C (C++, Fortran, etc) as C is to assembly. More to come here.

Data Flow, blocks that represent data manipulation, graphical, similar to block diagrams, lines represent data types,

Lastly, it is worth mentioning that LabVIEW is compiled, not interpreted. Among other things, this often means that when you are using new functionality, it will require some time for it to compile all the code, particularly the code that references hardware on your machine. Be patient!

1.1.3 Organization

This tutorial will be organized according to the needs of an introductory controls course. The goal is *not* to systematically introduce students to all of LabVIEW's capabilities. It is the author's view that this is simply an untenable goal leading to certain failure and confusion on the part of the students. Instead, we will focus on using simple examples, as they become useful, to illustrate new tools being used within LabVIEW. As the tutorial proceeds, it is assumed that less and less explicit instruction will be required, so some hints may simply lead the reader to look at a particular tool on his own and merely indicate that it is indeed a useful tool.

Chapter 2

Basics of Graphical Programming

Representing complex systems as block diagrams is already a common use of graphical representations. Textbooks use this regularly in control systems design, and the idea is largely to replace a mathematical representation of the entire system with a series of subcomponents connected together by various types of interconnections. This is the fundamental concept behind graphical programming as well, making it modular in design and flexible in most applications. Although there are certain types of computations that we will encounter where graphical programming is not a good choice, it will be largely to our advantage to program in this way.

2.1 Graphical Programming as Data Flow

Consider the following typical type of simple programming problem. Given a temperature in Fahrenheit F° , we want to compute the Celsius degrees C° . We want to use the formula

$$F = (212 - 32)/100C + 32$$

and

$$C = 100/(212 - 32)F - 32.$$

as expressions. In standard text languages, this might look something like:

```
float celsius(float fahrenheit);

main()
{
    float DegreeF=76;
    float DegreeC=celsius(DegreeF);
    printf("The Temperature is %f F and %f C\n", DegreeF, DegreeC);
}

float celsius(float fahrenheit)
{
    float celsius = (5.0/9.0)*(fahrenheit-32);
    return celsius;
}
```

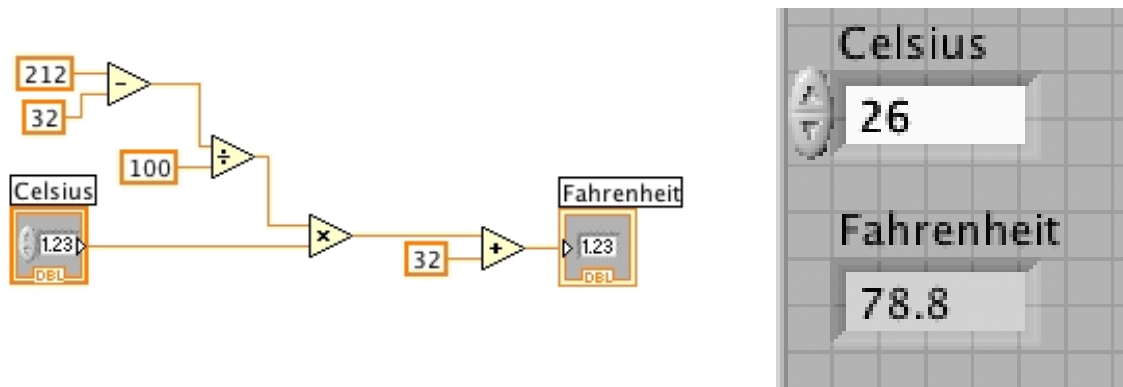


Figure 2.1: A LabVIEW-related graphic

Chapter 3

Simulation Module

3.1 Simulation in LabVIEW

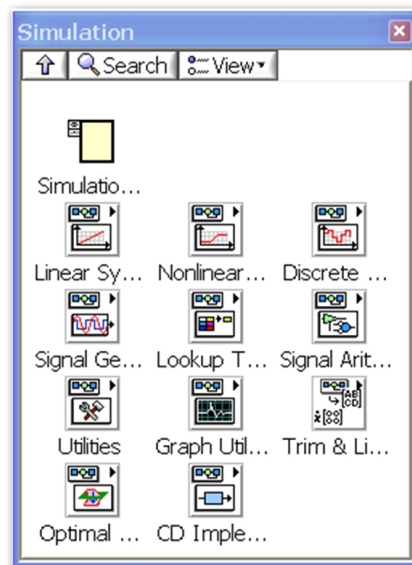


Figure 3.1: The Simulation Palette

The “simulation loop” will be one of the fundamental tools used in both simulating systems *and* running hardware-in-the-loop experiments. It can be obtained by opening up the simulation palette in the back panel (shown in Fig. 3.1), clicking on the upper left “simulation loop” icon, and dragging it into the back panel. It may then be stretched to any desired size. The result will look something like the loop seen in Fig. 3.2. All the things that need to execute at every time step in a simulation or hardware experiment should be

placed within the simulation loop. Everything else can be kept outside of it.

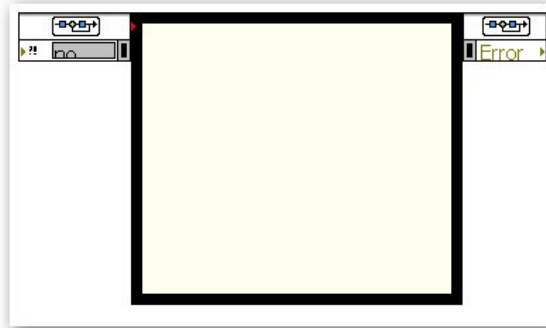


Figure 3.2: The Simulation Loop

The simulation loop determines the

1. integration algorithm used (e.g., Euler (fixed time step), Runge-Kutta, and variable time step algorithms);
2. Determines the step size both for fixed time step integration algorithms and for running hardware experiments;
3. length of time for the simulation;
4. timing parameters (involving the clock, an external clock, how fast to access said clock, etcetera).

All these options can be configured by right-clicking on the simulation loop and selecting Configure Simulation Parameters. The dialog box is shown in Fig.3.3

The palettes contain the other operations you may wish to use. These include:

1. Generating a signal. For instance, one can create a step-input from the Signal Generation palette. After placing the step-function in the Simulation Loop, double click on it. This will bring up the configuration window. Within the configuration window is a list of parameters. Clicking on any of the parameters will open “Parameter Information,” which allows you to modify the selected parameter (i.e. final value of step-input = 10).
2. Mathematical operations, such as summation and multiplying a signal by a gain, can be found in the Signal Arithmetic palette. Each of these can be individually configured—for instance, clicking on the summation block in the panel will allow one to configure it for “negative” feedback.



Figure 3.3: A LabVIEW Simulation Loop

3. Linear effects, such as time delay are found in the Linear Systems palette. (Note that within LabVIEW, a time delay is known as “transport delay.”)
4. Nonlinear effects such as saturation are found in the Nonlinear Systems palette.
5. Graphics: Plots can be generated using Graph Utilities. In general, you will want to use a sim-time waveform, but you may wish to use an XY graph if you want trace functionality.

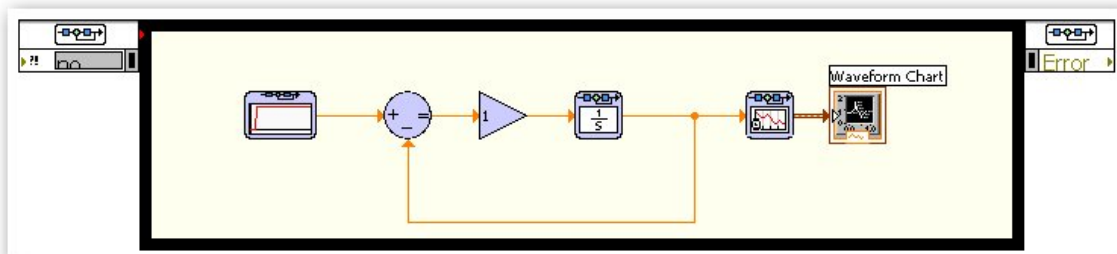


Figure 3.4: A simple feedback controller

3.1.1 Creating Sub-Systems in your Simulation

You will find that your code gets quite complicated as you get more functionality. If you want to replace part of your code with a subsystem block, just select the parts you want in the subsystem and go to *Edit* and select *Create Simulation Subsystem*. This will create a block that has the same inputs and outputs and the region you selected. You can then view the contents of the block by right clicking on it and selecting *Open Subsystem*.

More information on the simulation module can be found at [?].

3.2 MathScript—a command line utility

MathScript has much of the functionality of other command line scripting languages such as Matlab and MatrixX. Standard commands useful to control design include:

1. *tf* creates a transfer function;
2. *step* plots the step response of a transfer function or state space system;
3. *bode* plots the bode plot of a transfer function;
4. *rlocus* plots the root locus of a transfer function;
5. *nyquist* plots the nyquist plot of a transfer function.

Example code that might be illuminating includes:

Here are some different ways of doing Example 2.1 in the FPE textbook. You can get a step response entirely numerically, like the book does:

```
num=1/1000;
den=[1 50/1000];
sys=tf(num*500,den);
step(sys)
```

You can use variables:

```
m=1000;
b=50;
num=1/m;
den=[1 b/m];
sys=tf(num*500,den)
```

You can define the transfer function as a fraction (i.e., the way you would write it down). This is ultimately going to be preferable.


```

s=tf('s');
g=(500/m)/(s+b/m)
step(g)

```

Lastly, you can incorporate “.m” file code into your LabVIEW VI by using the MathScript block from the Structures palette. An example of this is given in Fig. 3.5. In order to do this, one must create the MathScript block, write the desired code, create an output of the block (by right clicking on the edge of the block), and finally define that output to be of the appropriate data type (in the case of Fig. 3.5 a “TF Object”, and in the case of Figs. 3.6, 3.7, and 3.8 a “SS Object”).

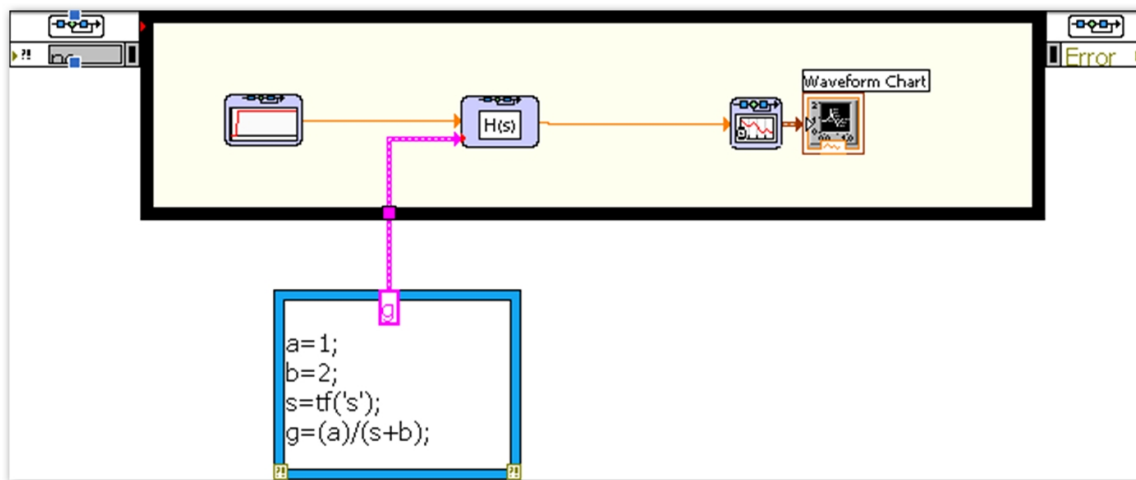


Figure 3.5: MathScript defining a transfer function in the simulation module

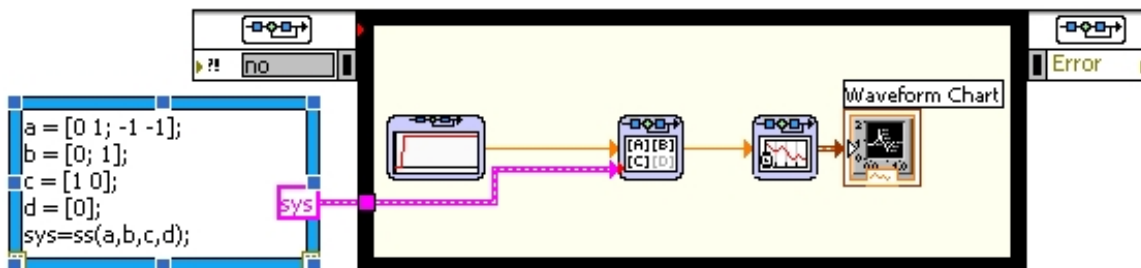


Figure 3.6: MathScript defining a state space system in the simulation module

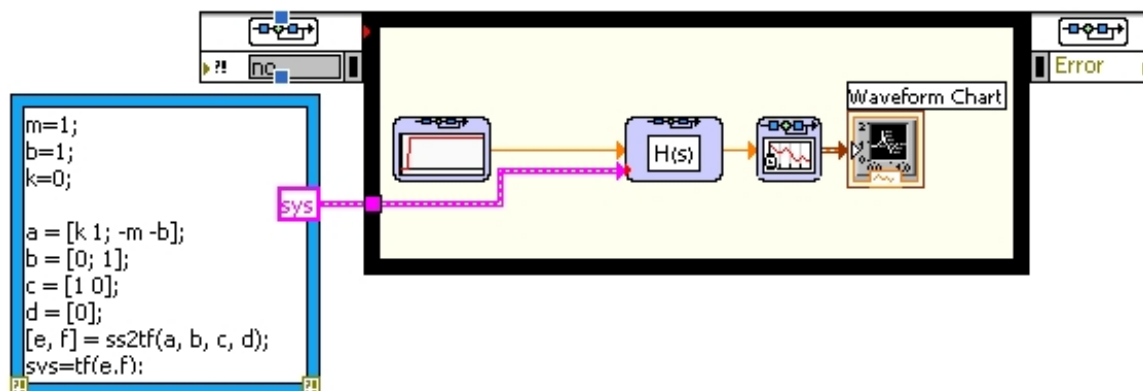


Figure 3.7: MathScript defining a state space system, converting it to a transfer function, and using it in the simulation module

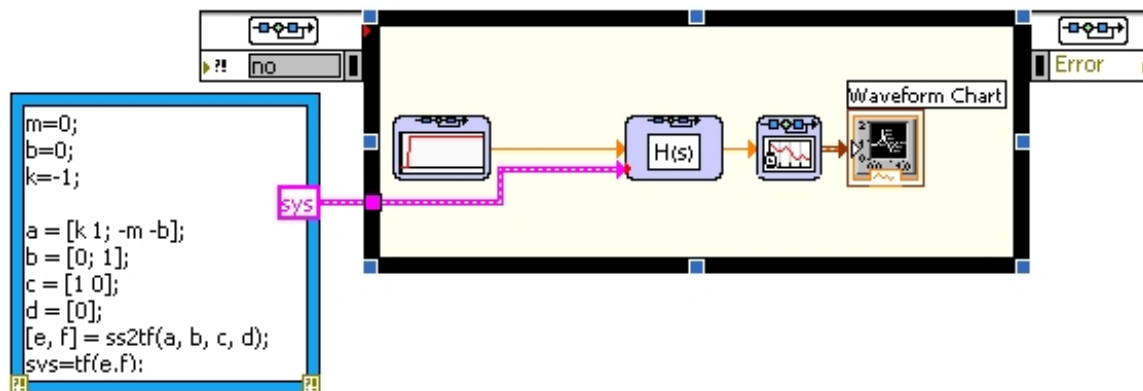


Figure 3.8: MathScript defining a degenerate state space system, converting it to a transfer function, and using it in the simulation module

3.3 Graphs in LabVIEW Simulation Module

Graphs are typically created using a sim-time waveform (in the simulation module), as seen in Fig.3.9. However, if you want trace functionality, the you should use the XY Graph (also in the simulation module). The back panel for this is seen in Fig. 3.10. Note that the XY Graph requires both time (which we generate in this case using a ramp function) and the actual signal, which are then plotted against each other. In order to plot them both, one must “bundle” the two signals together using a bundle block (located in the “Cluster and Variant” palette).

The cursor is created in the front panel (see Fig. 3.11) by right clicking on the graph and selecting *Properties*. Then select cursor, choose to add a cursor, and a cursor will show up

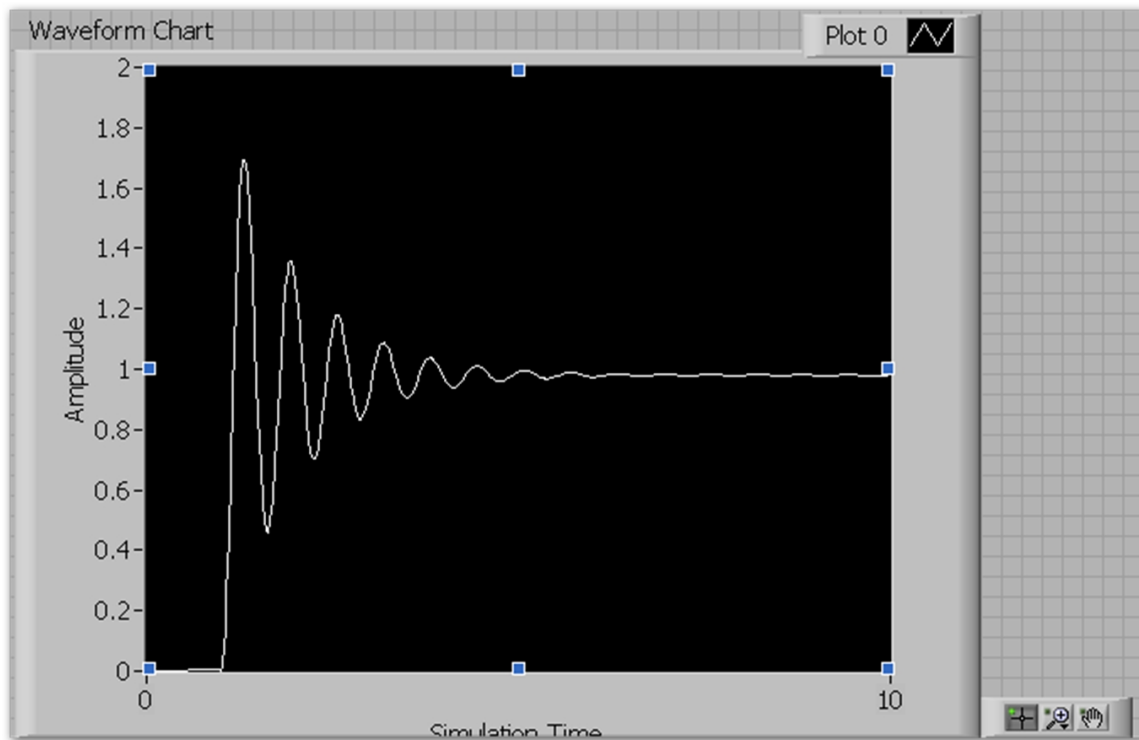


Figure 3.9: A Sim-Time Waveform

on the graph. Note that the cursor can only select actual data points—it will not interpolate. Therefore, you may wish to select a maximum step size in your integration algorithm that makes it move smoothly from point to point. (E.g., reduce the maximum step size.)

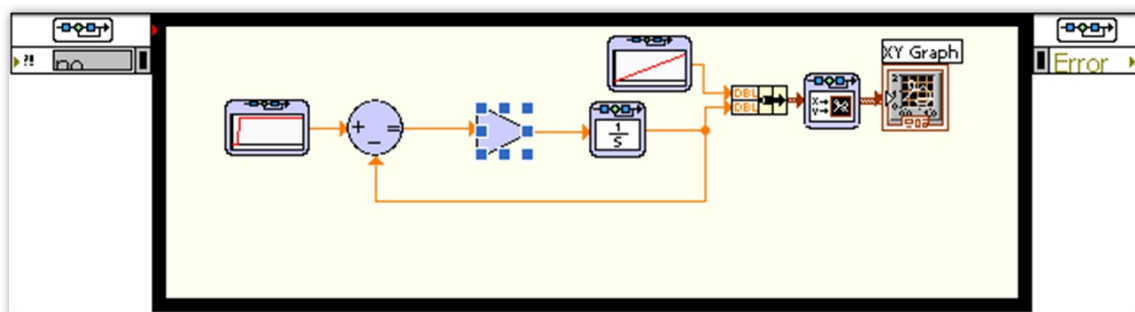


Figure 3.10: An XY Graph Back Panel

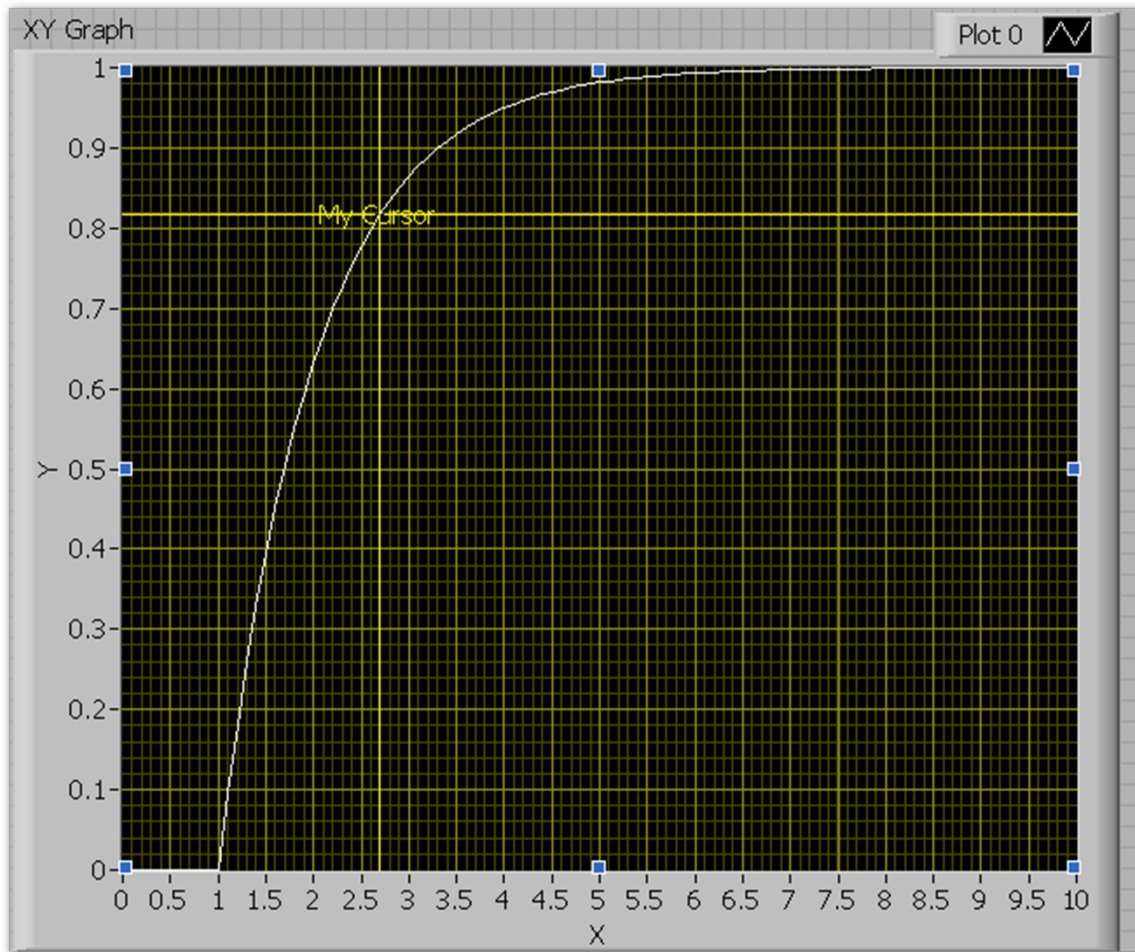


Figure 3.11: A XY Graph Front Panel with Cursor

3.3.1 Some Notes on Graphing in LabVIEW

1. There are two distinct types of XY Graphs. One is the Simulation Module “Buffer XY graph” (found in the Simulation Module Palette under “Graph Utilities”) that *can only be used in the simulation module*. The other is the standard XY graph that can be found under the “Graph” palette of the *front panel*. (I.e., you find it by right-clicking on the front panel, select the “Graph” palette, and then click on “XY Graph.”)
2. The default amount of data a Waveform Chart in the Simulation Module displays is 1024. If your time steps are too small in a simulation or experiment, you may not get all your data. Hence, if this happens you should change the “Chart History Length” to a larger number (by right clicking on the graph).

3. Lastly, the time axis sometimes gets set to Day/Year format. If this happens, go into “Properties” and under “Format and Precision” set Type to SI notation.

3.4 LabVIEW Saving and Reading

This tutorial is to give you a sense of how to save and read data within LabVIEW. Saving data is reasonably straight forward. You create a VI, and use a “Collector” to collect all the data during the simulation/experiment. Then you must use “Unbundle By Name” and put all the data you want to save into the same array by using “Build Array.” Lastly, the “Write File To Spreadsheet” utility will allow you to write the file in a convenient format. (All this is shown in Fig.3.12.) Note that you *must* allow the simulation to finish or data will not be written to the file. Also note that LabVIEW allows you to plot multiple plots on the same graph, as shown.

Now, reading data is similarly straight forward. Assuming that you have saved the data as above (in a spreadsheet format), you can use the “Read From Spreadsheet” utility. The data will be in an array format already, but you must use the “Index Array” to select the data. Figure 3.13 shows the use of the “Index Array” block to select the first (“0”th) and second (“1”st) signals from the data. These can then either be used or bundled together and plotted.

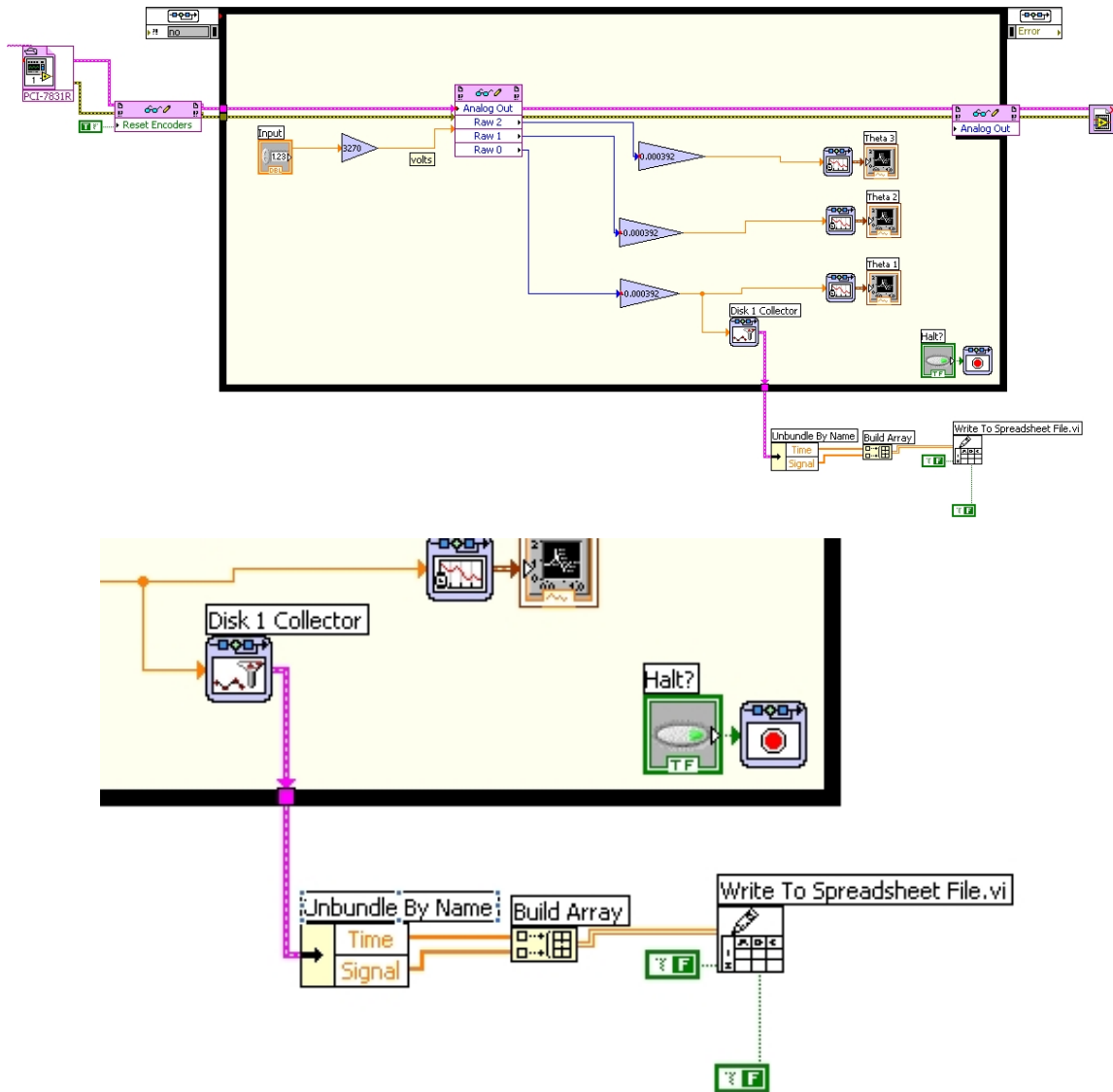


Figure 3.12: (top) A VI that saves data from an FPGA, (bottom) an enlargement of the section of code that saves data

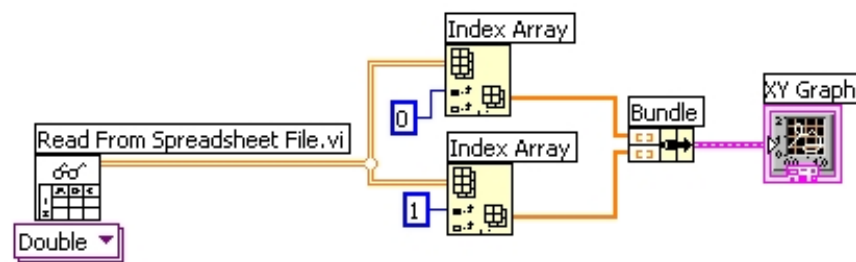


Figure 3.13: A VI that reads data

Chapter 4

Analysis

This chapter introduces the basic computational tools used for design and analysis purposes. These include frequency domain techniques (Root Locus, Bode Plot, and Nyquist Plot) as well as state-space techniques (stability, pole-placement, estimator design, LQR). We end with a short discussion of digital control.

4.1 Frequency Domain Analysis

4.1.1 Root Locus

LabVIEW can plot root loci in your VI using the **CD Root Locus** block, which can be found on the Block Diagram palette under **Control Design and Simulation**→**Control Design**→**Dynamic Characteristics**. The advantage of using LabVIEW's root locus is that you can plot the root locus for your controller on the VI right next to your simulation or experimental results. Some things to know:

1. Once you have placed the block, right click on the “Root Locus Graph” output and select **Create**→**Indicator** to create a plot on the Front Panel.
2. You must connect your model to the block's model input. The model can be a state-space, transfer function, or zero-pole-gain model.
3. You can plot the pole locations for specific gains by creating an array of the gains and connecting them to the “Gain” input. The VI in Fig. 4.1 will show the poles for $K = 1$ and $K = 100$.
4. You can draw your model equations using the **Control Design and Simulation**→**Control Design**→**Model Construction**→**CD Draw State Space Equations**. Right click on

the block's "Equations" output and create an indicator to draw the system model. Alternatively, you can convert your model to a transfer function and draw it using the CD Draw Transfer Function block.

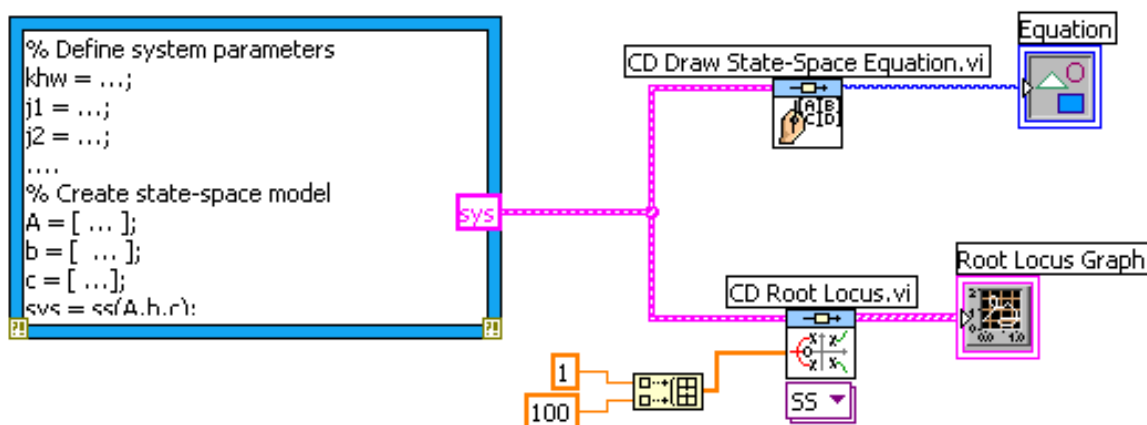


Figure 4.1: This VI will plot the root locus along with the poles for $K = 1$ and $K = 100$. The state space equations for the model will also be drawn.

LabVIEW also provides an interactive root locus plotter. You vary the gain and directly see how the poles of the system move. Invoke the interactive root locus plotter using the command `rlocfind(SYS)` in a MathScript node where `SYS` is the name of your model. Things you might want to know:

1. You might have to change the feedback type to positive depending on how you juggled the signs around in your model.
2. When you close the interactive root locus, the `rlocfind` function returns the selected gain. You can output the gain and use it in your simulation to test it. See Fig. 4.2.

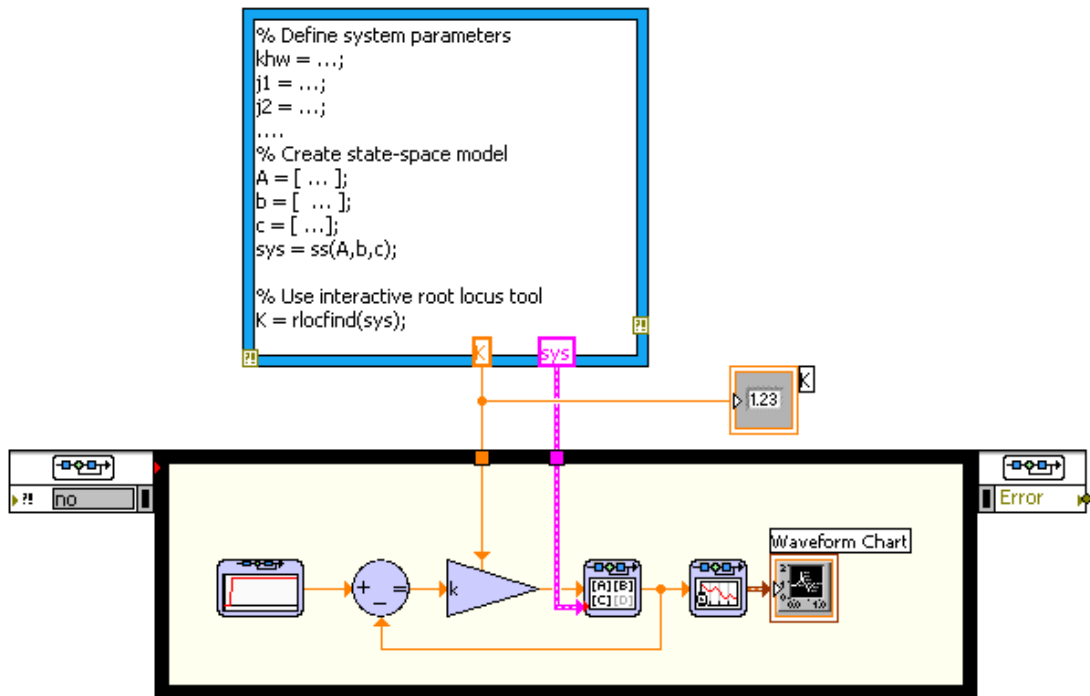


Figure 4.2: This VI uses the interactive root locus plot to choose a gain and simulates the step response for that gain.

4.1.2 Bode

4.1.3 Nyquist

4.2 State Space Analysis

4.2.1 Stability

4.2.2 LQR

4.3 Digital Control

Chapter 5

LabVIEW 8.2 Installation

5.1 Image Installation

The installation files for LabVIEW 8.2 and the required modules will be copied to the hard drive so that the LabVIEW development platform can be quickly reinstalled when the computer is re-imaged.

1. Create a new directory: `C:\LabVIEW 8.2 Installation`
2. Insert the installation disk.
3. Copy all of the files on the disk to `C:\LabVIEW 8.2 Installation`
4. Remove the installation disk.

An installation image can now be created from the hard drive.

5.2 Installation

This subsection assumes that the computer has been properly imaged and has the directory `C:\LabVIEW 8.2 Installation` that contains all of the installation files. LabVIEW and the required modules will now be installed. If an FPGA board will be used on the machine, the FPGA 8.20 and NI-RIO 2.1 modules must be installed. The installation can be done manually or using the included batch file. If an NI FPGA board is already installed in the computer, it does NOT have to be removed for installation.

5.2.1 Automatic Installation

For automatic installation, run `C:\LabVIEW 8.2 Installation\setup.bat`. The file will install:

1. LabVIEW 8.20
2. LabVIEW FPGA 8.20
3. National Instruments RIO 2.1
4. Simulation Module 8.20 Beta
5. Control Design Toolkit
6. System Identification Toolkit 3.0
7. LabVIEW 8.20 Patches for Control Design Toolkit 2.1
8. Todd Murphey's Torsional Plant FPGA VIs

The installation takes a significant amount of time. Be patient before deciding the installation has locked up and rebooting the computer. Log files of the installation will be created in `C:\LabVIEW 8.2 Installation`. Once the batch file has finished the installation, reboot the computer and continue to the next subsection.

5.2.2 Manual Installation

1. Run `C:\LabVIEW 8.2 Installation\LabVIEW 8.20 English\setup.exe`
2. (Optional) To install LabVIEW FPGA 8.20, run `C:\LabVIEW 8.2 Installation\LabVIEW FPGA 8.20\setup.exe`
The FPGA installation will prompt you to install NI-RIO. Select the directory `C:\LabVIEW 8.2 Installation\NI-RIO 2.1\NIRIO`
3. Run `C:\LabVIEW 8.2 Installation\Simulation Module 8.2 beta\setup.exe`
4. Run `C:\LabVIEW 8.2 Installation\Control Design\setup.exe`
5. Run `C:\LabVIEW 8.2 Installation\System Identification Toolkit 3.0 \setup.exe`

There are two patches for the Control Design Toolkit that should be installed:

1. Unzip the file `C:\LabVIEW 8.2 Installation\patches\CDT21Err20111Fix.zip` to `C:\Program Files\National Instruments\LabVIEW 8.2\`. If prompted, select "Yes" to overwrite the directory `vi.lib`.

2. Unzip the file `C:\LabVIEW 8.2 Installation\patches\CDT21binFileFix.zip` to `C:\Program Files\National Instruments\`. If prompted, select “Yes” to overwrite the directory `Shared`.

Finally, copy the directory `C:\LabVIEW 8.2 Installation\code\` to `C:\Program Files\National Instruments\`. This file contains FPGA VI's that can interface with the torsional plant hardware.

5.3 Installing the FPGA

If the NI-FPGA board is not installed, turn off and unplug the computer, insert the board into an empty PCI slot, plug in and turn on the computer.

When Windows has started, a **Found New Hardware...** dialog should pop up. Let the hardware wizard automatically find and install the drivers.

5.4 Configuring LabVIEW

When LabVIEW is first opened, it must be activated.

1. Open LabVIEW 8.2
2. An “Evaluation License” dialog will open. Select “Activate.”
3. Select “Automatically activate through a secure internet connection.”
4. Enter the serial number M21X98212
5. Enter Todd Murphey for the first and last name, and CU Boulder ECEE Dept. for organization.
6. Uncheck the registration box and select “Next”
7. Select “Next”
8. Select “Finish” once the activation is complete.
9. If the Windows firewall dialog opens, select “Unblock”
10. The “Prompt for Mass Compile” dialog will open. Select “Mass Compile Now” and wait for the compilation to finish.
- 11.

5.5 Using the NI-7831 FPGA with the ECP Model 205a Torsional Plant

This subsection describes how to use Todd Murphey's included VIs to interface with the ECP Model 205a Torsional Plant. First, the hardware must be properly connected.

1. Connect a NI SCB-68 breakout box to the MIO-C0 port on the NI-7831 FPGA board.
2. Set SCB-68 configuration switches to use the NI-7831:
 - (a) Switch 1 Left
 - (b) Switch 2 Left
 - (c) Switch 3 Down
 - (d) Switch 4 Up
 - (e) Switch 5 Up
3. Using the double-wire, connect the Plant Drive Power connector to the Power Amplifier MOTOR connector.
4. Using the single-wire, connect the Plant Feedback connector to the SCB-68 breakout box. The connections between the single-wire outputs and the SCB-68 are as follows:
 - (a) Red - Pin 1 (+5V)
 - (b) Black - Pin 2 (DGND)
 - (c) Yellow - Pin 36 (DIO0)
 - (d) White - Pin 37 (DIO1)
 - (e) Blue - Pin 38 (DIO2)
 - (f) Green - Pin 39 (DIO3)
 - (g) Brown(?) - Pin 40 (DIO4)
 - (h) Orange - Pin 41 (DIO5)
5. Connect the Power Amplifier DAC input to SCB-68 Pin 55 (AO0)
6. Connect the Power Amplifier DAC/ input to SCB-68 Pin 21 (AOGND0)
7. Check that the amplifier power switch is turned off.
8. Plug in the power amplifier.

5.5. USING THE NI-7831 FPGA WITH THE ECP MODEL 205A TORSIONAL PLANT33

The hardware setup should be complete. The LabVIEW FPGA code can now be compiled and tested:

1. Open LabVIEW.
2. Open the file `C:\Program Files\National Instruments\code\4638fpgaproject.lvproj`. If the file fails to open, create a new project:
 - (a) Left click on “Empty Project.”
 - (b) Save project.
 - (c) Right click on “My Computer” and select “New - > Targets and Devices.”
 - (d) Under “Existing target or device” select “FPGA Target - RIO0:INSTR (PCI-7831R).”
 - (e) In the project, right click on “FPGA Target...” and select “Add File...” and add “reading_encoders.vi” to the project.
 - (f) Save project.
 - (g) Right click on “FPGA Target...” and select “New FPGA I/O”. Add Analog Output: AO0, and Digital Line Input and Output Connector 0: DIO0, DIO1, DIO2, DIO3, DIO4, DIO5.
3. In the project, expand **FPGA Target** and double click on **reading_encoders.vi**.
4. Hit Ctrl E to see the back panel (where the code is) to make sure that there are no broken wires in the code. If there are, contact me.
5. In the project, right click on **reading_encoders.vi** and select “Compile.” If you get an error saying that it cannot contact the compile server, hit retry.
6. Once the VI has compiled, go to the front panel, hit run, and verify that rotating the ECP unit counterclockwise causes all the counts to go up. If not, double check all connections. The power amplifier should not have to be turned on to use the encoders. They receive power through SCB-68 pin 1. If the VI still does not respond, this will have to be fixed in the FPGA code.
7. Close the FPGA project.

5.6 Using the `reading_encoders.vi` FPGA interface VI.

In addition to the FPGA code, the installation includes two standard VI's that use the FPGA VI. The first VI, `targetcode.vi` continuously reads the FPGA output to plot the angle of each disk.

1. Open `C:\Program Files\National Instruments\code\targetcode.vi`
2. Press Ctrl-E to open the back panel.
3. Right click on the FPGA Node labeled **NI-7831R** near the left side of the panel and choose "Select Bitfile..." The bitfile is the object file that was compiled from the FPGA VI. By assigning a bitfile to the FPGA node, the client VI can use FPGA VI without having to recompile the FPGA code.
4. Under the **Bitfiles** subdirectory, select the file `4638fpgavi.lvbit`.
5. Check the VI to make sure that all wires are now connected.
6. Run the VI and rotate the disks. The plots should display the angles of each disk.
7. Press the "Halt?" button to stop the VI.

The second VI implements a closed-loop PID controller to control the angle of the disks.

1. Open `C:\Program Files\National Instruments\code\targetcode.vi`
2. Press Ctrl-E to open the back panel.
3. Right click on the FPGA Node labeled **NI-7831R** near the left side of the panel and choose "Select Bitfile..." The bitfile is the object file that was compiled from the FPGA VI. By assigning a bitfile to the FPGA node, the client VI can use FPGA VI without having to recompile the FPGA code.
4. Under the **Bitfiles** subdirectory, select the file `4638fpgavi.lvbit`.
5. Check the VI to make sure that all wires are now connected.
6. Run the VI.
7. Enter a small **NEGATIVE** gain for the proportional controller.
8. Turn on the power amplifier. The current FPGA VI does not have any safeguards to prevent the system from becoming unstable. Keep your fingers near the power button.

9. Gently disturb the plant. The controller should apply inputs to return the disks to the correct angle.
10. Turn off the power amplifier.
11. Press the “Halt?” button to stop the VI.

5.7 Notes

Chapter 6

Torsional Plant System Identification

Objective

We provide several LabVIEW VIs that can be used to quickly collect and analyze data from a plant to determine model parameters. This section provides instructions for using the VIs and presents results from several ECP torsional plant units.

6.1 System Identification

The system identification problem is to create an accurate dynamical model of the system. The first step is to define the equations of motion for the plant.

The plant consists of the bottom, middle, and top disks with rotational inertias J_1 , J_2 , and J_3 and angles θ_1 , θ_2 , θ_3 . Each disk has a viscous dissipation force with damping coefficients c_1 , c_2 , and c_3 . There are torsional springs connecting the bottom and middle disks and the middle and top disks with respective spring constants k_1 , k_2 . The power amplifier and motor are treated as a single gain, k_h , with no dynamics. The resulting state space model is

$$\dot{x} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -\frac{k_1}{J_1} & \frac{k_1}{J_1} & 0 & -\frac{c_1}{J_1} & 0 & 0 \\ \frac{k_1}{J_2} & -\frac{k_1+k_2}{J_2} & \frac{k_2}{J_2} & 0 & -\frac{c_2}{J_2} & 0 \\ 0 & \frac{k_2}{J_3} & -\frac{k_2}{J_2} & 0 & 0 & -\frac{c_3}{J_3} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k_h}{J_1} \\ 0 \\ 0 \end{bmatrix} u \quad (6.1)$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} x \quad (6.2)$$

To complete the model, the values of the model parameters must be determined.

6.1.1 Rotational Inertia

The inertia parameters can be calculated from known quantities. The ECP manual provides the rotational inertia of an unloaded disk (0.0019 kg-m²), the mass of each weight (0.5 kg), and the radius of a weight (2.5 cm). By treating the weights as cylinders, the rotational inertia of a disk loaded with N weights can be found:

$$J = J_{disk} + \sum_{i=1}^N \left(\frac{1}{2}mr^2 + mR_i^2 \right) \quad (6.3)$$

where r is the radius of a weight and R_i is the distance from the weight to the center of the disk. For the bottom disk, rotational inertia of the motor-drive assembly can be included using the value from the ECP manual (0.0005 kg-m²).

6.1.2 Spring and Damper Coefficients

The spring and damper coefficients are most easily found by by clamping the middle disk to create two independent second-order systems. The transfer function for each disk in this configuration is

$$\frac{Y(s)}{U(s)} = \frac{1/J}{s^2 + (c/J)s + k/J} = \frac{K}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (6.4)$$

where ω_n , ζ , and K are the standard second order plant parameters. The impulse response of a second order system is an exponentially decaying sinusoid given by the equation:

$$y(t) = y_0 e^{-\sigma t} \sin \omega_d t \quad \sigma = \zeta\omega_n \quad \omega_d = \omega_n \sqrt{1 - \zeta^2}$$

where y_0 is the initial angle of the disk.

Measuring the impulse response is equivalent to choosing non-zero initial conditions for the plant and measuring the unforced response of the plant. Any number of methods can be used to determine the frequency, ω_d , and the rate of decay, σ from the data. Once these parameters are known, the spring constant and damping ratio of the corresponding disk can be calculated. It is assumed that $c_3 = c_2$.

6.1.3 Hardware Gain

The hardware gain, k_{hw} is the ratio of the applied torque and specified voltage. It represents the combined dynamics of the power amplifier and motor, and is assumed to be constant for the lab course.

To find k_{hw} , the plant is reduced to a different second order system by unclamping all disks and removing the weights from the top and bottom disks. For further accuracy, the top and bottom disks can also be removed. The resulting transfer function is

$$\frac{Y(s)}{U(s)} = \frac{k_{hw}/J}{s(s + C/J)}$$

To determine the gain, we apply an input $U(s)$ to the plant and record the system response. The same input is used to simulate the plant by assuming $k_{hw} = 1$ along with the parameters found earlier. The actual gain is found by comparing the two responses:

$$\frac{Y_m(s)}{Y_s(s)} = \frac{\frac{k_{hw}/J}{s(s+c/J)}U(s)}{\frac{1/J}{s(s+c/J)}U(s)} = k_{hw}$$

where Y_m and Y_s are the measured and simulated responses, respectively. Rather than compare the outputs for all time, the final steady state values are used.

6.2 VI Implementations

Four LabVIEW VI's are provided for doing system identification:

System ID Recorder Records the unforced response to non-zero initial conditions to determine C and K .

System ID Analyzer Analyzes data from the system ID recorder.

Hardware Gain Recorder Applies a pulse input to the system and records response to determine k_{hw} .

Hardware Gain Analyzer Analyzes data from the hardware gain recorder.

The VI's were designed to conveniently and quickly capture and analyze a large number of data sets. They allow the user to quickly determine parameters, compare simulated responses to actual data, and investigate nonlinearities in the system.

6.2.1 System ID Recorder

The System ID Recorder VI is implemented in `system_id_logger.vi`. This VI records the unforced response of a disk and saves the results to a file. To use the VI, follow these steps:

1. Disconnect the motor power or turn off the power amplifier.

2. Securely clamp the middle disk to the frame.
3. Place weights on the disk to be used. Calculate the disk inertia for the weight configuration.
4. Select the appropriate disk from the pull-down menu on the VI front panel.
5. Enter an appropriate recording time on the VI front panel. The time should be long enough for the disk to come to rest after an initial push, plus a little extra time for you to push the disk.
6. Start the VI. Wait until the FPGA is initialized and the VI begins plotting data.
7. Gently move the disk and quickly release it. The disk should not be turned more than 40 degrees or the spring could be damaged.
8. When the VI finishes, a dialog box will ask you where to save the file. Select a directory and enter a new filename, or select a file that was created using the same VI and plant configuration. The VI will remember your choice and append future data to the file. If you want to write to a new file, either clear the filename control on the front panel, or enter the new filename.
9. You can run the VI repeatedly to gather more data from different initial conditions. To use the data effectively with the analyzing VI, the same weight configuration should be used for all data runs saved in a file. Create a new file if the number of weights are changed.

Once you have recorded enough data, you can analyze it to find the spring and damper coefficients using the analyzer VI.

The data is saved to file using LabVIEW's **Write to Spreadsheet** Sub-VI. The file is an ASCII text file where each data set consists of two rows. The first is the time relative to the start of the VI and the second is the measured position of the disk at that time. These files can easily be imported into Excel, MATLAB, or another LabVIEW VI.

6.2.2 System ID Analyzer

The System ID analyzer is implemented in `system_id_analyzer.vi`. When you run the VI, a dialog will ask you choose the data file. Only use files created with the System ID Recorder.

This VI uses recorded unforced responses and a calculated rotational inertia to estimate the spring and damper coefficients for a disk. The parameters are plotted with respect to their data sets initial conditions to provide some insight into non-linearities in the system. The VI will also simulate the system with the estimated parameters for comparison against the recorded data.

Implementation

The VI first selects two peaks from the data. The first peak is chosen by taking the first peak after the global minimum of the data. As long as the disk was only pushed to one direction to create the initial conditions, this tends to select the first or second peak of the unforced response. This point (t_0, y_0) is considered the initial condition of the data. The second point (t_1, y_1) is the last peak that is more than 5% of the initial condition. The damped frequency and decay rate are calculated from these two points:

$$\omega_d = \frac{2\pi(N-1)}{t_1 - t_0} \quad \sigma = -\frac{\ln y_0 - \ln y_1}{t_1 - t_0} \quad (6.5)$$

where N is the number of peaks between and including the two chosen peaks. These are converted to the damping ratio and natural frequency of the system:

$$\omega_n = \sqrt{\omega_d^2 + \sigma^2} \quad \zeta = \frac{\omega_n}{\sigma} \quad (6.6)$$

The second order parameters are used to calculate the spring and damper coefficients according to Equ. 6.4.

$$c = 2J\omega_n\zeta \quad k = J\omega_n^2 \quad (6.7)$$

6.2.3 Hardware Gain Recorder

The Hardware Gain Recorder is implemented in `hardware_gain_logger.vi`. This VI applies a 1 second wide pulse to the plant and records the response. Follow these steps to use this VI:

1. Make sure all disks are unclamped and the bottom disk is completely free to rotate.
2. Remove all weights from the top and middle disk. For additional accuracy, remove the disks themselves from the plant.
3. Place weights on the bottom disk and calculate the disk inertia.
4. Connect the power amplifier to the motor and turn on the power amplifier.

5. Enter the desired pulse voltage on the VI front panel.
6. Enter an appropriate recording time on the VI front panel. The time should be long enough for the system to come to rest.
7. Run the VI. After approximately 2 seconds, the pulse will be applied and the system will spin and slowly come to rest.
8. When the VI is finished, a dialog box will ask you where to save the file. The filename will be saved and used for future data sets. Use the same file for all data runs with the same weight configuration. The recording time and voltage can change within the file.
9. Once you've chosen a filename, you can use LabVIEW's "Run continuously" function to capture several data runs at a single value.
10. Be sure to capture data for different pulse voltages to see how the gain changes with the applied voltage.

The data is saved to file using LabVIEW's **Write to Spreadsheet** Sub-VI. The file is an ASCII text file where each data set consists of three rows. The first is the time relative to the start of the VI, the second is the measured position, and the third is the input signal. These files can easily be imported into Excel, MATLAB, or another LabVIEW VI.

6.2.4 Hardware Gain Analyzer

The hardware gain analyzer is implemented in `hardware_gain_analyzer.vi`. When you run the VI, a dialog will ask you choose the data file. Only use files created with the Hardware Gain Recorder.

This VI uses recorded pulse responses to estimate the hardware gain of the plant. The hardware gain is plotted against the pulse voltage to provide insight into non-linearities in the system. The VI also simulates the system with the estimated gain for comparison against the actual data.

Implementation

The VI determines the voltage of the applied pulse and the final steady state value of the system from the recorded data. A simulation of the system is run using the specified rotational inertia and damping coefficient with $k_{hw} = 1$. The hardware gain is determined as

$$k_{hw} = \frac{y_{exp}(t_f)}{y_{sim}(t_f)} \quad (6.8)$$

where t_f is the time of the last recorded measurement.

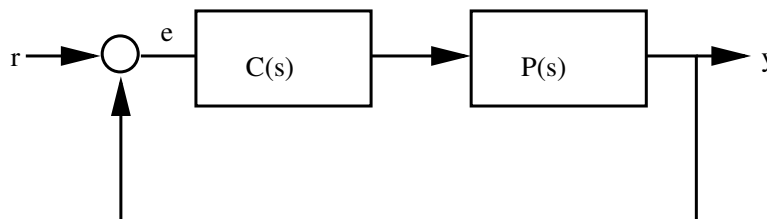
Appendix A

Lab #1: Introduction to Digital Simulation in LabVIEW

Objective

The purpose of this lab is to become familiar with using the graphical programming language provided by LabVIEW (or other similar languages such as SIMULINK) for simulation and implementation of digital control systems. The main steps in this lab will be the following. First, you will simulate a linear model of the torsional disk system with only one disk and that is using a proportional controller. You will then include the effects of saturation and time delays in your model. Lastly, you will implement a proportional controller on your simulated system. Before doing any of this, you will need to be comfortable with the LabVIEW programming environment, so you may wish to look at the tutorial that will be available on <http://moodle.cs.colorado.edu>. You may alternatively wish to go to www.ni.com and look at one of their interactive tutorials. Lastly, a PDF of a user's guide for LabVIEW should be on the lab computers at **Start>>Programs>>National Instruments>>LabVIEW 8.2>>Search the LabVIEW Bookshelf.pdf**.

A.1 Tasks



In this lab we will be considering a second-order mechanical system. For now, assume the system is linear and can be represented as a feedback loop (seen in Fig.A.1). Assume that $C(s) = K_p$ and $P(s) = \frac{K}{s^2 + 2\zeta\omega s + \omega^2}$. This will be the *idealized* plant for the torsional disk system with only one disk. For this lab, use the parameter values $K = 1$, $\zeta = 1.25$, and $\omega = 2$.

A.1.1 Task #1—Step Response

Examine the response of the system to a step response of 10 for 30 seconds. (You may shorten this length of time if it will be helpful.) Find the maximum value of K_p that results in 0% and 50% overshoot. Record how long it takes to get within 2% of the desired state (the 2% *settling time*) and how long it takes to get within the *steady state error* for both cases. Lastly, what are the *units* in this block diagram? Label each wire of the block diagram with its physical units. What does a step response of “10” mean physically?

For each case, turn in a plot that includes the *output*, the *step input*, and the *controller* input. (These can typically be copied directly from LabVIEW to MS Word. Plot control can be obtained by right clicking on the plot and choosing options.)

A.1.2 Task #2—Actuator Saturation

Although we model many systems as being linear, most are actually nonlinear. One of the most common nonlinearities that one can encounter is a *saturation*. All actuators have saturations because they cannot actually provide infinite force or torque. Use the LabVIEW saturation block to insert a saturation on the control in your simulation. Start with an step reference of size 1, and create a saturation of -200 to 200 on the output of your control block, and then decrement it until you start to see the effects in your simulation.

Using the same gains determined in Task #1, record the 2% settling time and the steady state error for a step size of 1. Then do so again for a step size of 10. How does this change your results? Obtain the same plots as requested in Task #1.

A.1.3 Task #3—Time Delay

Time delays are another problem in many control systems. Insert a time delay in the feedback loop to represent processing time for the sensors. At what time delay do you start seeing what you would deem “significant” difficulties?

A.1.4 Task #4—“Best” K_p

What do you think is the “best” K_p for this system if there are saturations and/or time delays? Defend your answer using the plots and data you have accumulated during the rest

of the exercise.

A.1.5 Task #5—Dependence on ω

What happens if you change ω ? In particular, what happens if you change it to 0 or to 10? How does this affect your performance/stability/etc?

A.2 Things You May Want To Know

All of the vi's that you need to complete this lab are part of LabVIEW's Simulation Module. Hopefully, the LabVIEW tutorial from class has been sufficient. However, a copy of the Simulation Module manual is available online at www.ni.com. Searching "Simulation Module" within this site will bring you to the pdf manual. Alternatively, you may wish to view Finn Haugen's website at <http://www.techteach.no/publications/labview/>.

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

Appendix B

Lab #2: Digital Simulation of Torsional Disk Systems in LabVIEW

Objective

The purpose of this lab is to increase your familiarity with LabVIEW, increase your mechanical modeling prowess, and give you simulation models that you can use to verify your designs before testing them on hardware.

The model you write today will allow you to simulate what the Torsional Disk System (TDS) will do in response to a given input. However, this response is dependent on the value of certain parameters for the system, such as the rotational inertia of the disks, the torsional spring constants, and the dissipation due to bearings and gears. These parameter values will have to be estimated using a process called *system identification*. (You will do this heuristically in this lab and will do it more formally in Lab #3.) Although system identification can involve no modeling whatsoever, it is almost always easier to do it when one is only trying to estimate a finite number of parameters. Hence, it is generally better to think of modeling and system identification as being inter-dependent. Once you have a model that is reasonably accurate, you will finally be in a position to do some actual control design (which will start in Lab #4).

It is worth noting that this is roughly the cycle that you would go through if you were doing an actual industrial project. This cycle is generally 1) Model, 2) Verify Model Experimentally, 3) Design Using Model, and finally 4) Test Control Design Experimentally. In general, no one would allow you to try something out experimentally until you could show that the simulations indicate it should work (or at least this is typically true).

Lab Overview

First, you need to find a model for the torsional disk system. As you know, it has three disks with different inertias and dissipation, and a torsional rod that can be twisted to

some degree, but has a linear torsional spring constant associated with it that (along with dissipation in the system) relaxes each disk back into its original orientation. It is driven by a DC motor that you can think of as directly applying a torque to the system, where the torque is scaled by some factor X .

Second, you will need to show that you can create a simulation for one disk, two disks, or all three disks. You can be as clever as you want in terms of how you program this. However, I would suggest using a MathScript node in the simulation module, such as Fig. 3.5.

Lastly, you will investigate what goes right/wrong with various feedback control approaches. This is largely intended to give you an appreciation of why we think of control *design* as actually being quite important!

B.1 Pre-Lab Tasks

The pre-lab requirements are roughly as follows:

1. Read this entire document
2. Prepare a model for the Torsional Disk System. This will be a 2nd-order differential equation of three variables. Be prepared to discuss your modeling with your teammates. (It is unlikely that you will all arrive with the same model.)
3. Based on simulations, guess the parameters for the Torsional Disk System.

B.2 Tasks

The torsional disk system has three disks of inertia (J_1, J_2, J_3) (due to the disks themselves and the masses we affix to the disks), damping coefficients c_1, c_2, c_3 (due to friction in the bearings and other components supporting the disks), a torsional rod connecting all three with torsional spring constants k_1 and k_2 in between the disks, and a torque scaling constant k_m that scales signals (voltage) from the FPGA to the output torque of the motor. When doing system identification, these are the parameters you will need to be able to identify in the next lab. However, in order to identify them, you need a model describing how the system behavior changes as one modifies the parameters.

B.2.1 Task #1—Mechanical Model

Task: Based on what you have learned in class, find a model for this system. Let the configuration variables of the bottom disk, middle disk, and top disk be $(\theta_1, \theta_2, \theta_3)$ respectively (with their positive directions measured by the right hand rule with the z-axis going up), the moments of inertia be J_1, J_2, J_3 , the viscous dissipation coefficients be c_1, c_2, c_3 (i.e., with

$\tau = -c_i \dot{\theta}_i$), and the spring constants between disk 1 and 2 be k_1 (i.e., $\tau = -k_1(\theta_1 - \theta_2)$) and between disk 2 and 3 be k_2 (i.e., $\tau = -k_2(\theta_2 - \theta_3)$). Be sure to be careful with your sign conventions! Lastly, let the input torque from the motor be T .

Task: What are the input, state, and output choices we have for this system?

Task: How might you check to see if your sign conventions are correct?

Task: What do you think are reasonable values of each of these values? You should be able to approximate this analytically for the inertia (for a given set of masses attached to the disk). The other parameters you will have to guess, although you should defend your answer (partially by noting whatever units you are using). As you think about defending your choice of parameters, you should think about what these choices would mean for the actual physical system that we have in the lab. For instance, you saw me twist the two disks relative to each other, so you know that the spring constant cannot be too high. Thinking about physical units might also be helpful in considering your guesses. What else do you know? You will have a chance in a moment to update your answer based on simulations.

Variable	Value
J_1	
J_2	
J_3	
c_1	
c_2	
c_3	
k_1	
k_2	

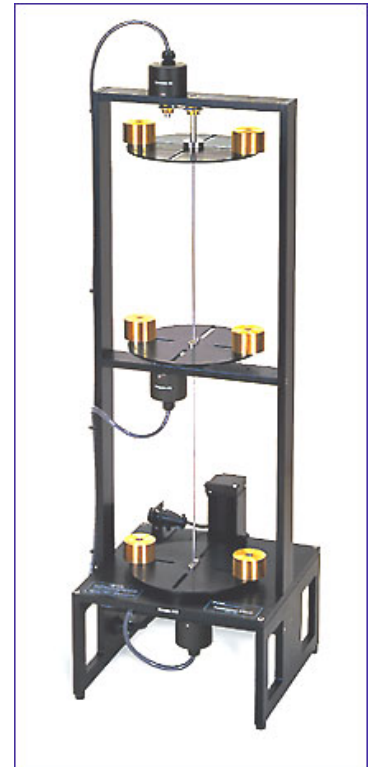


Figure B.1: The ECP Torsional Disk System.

Task: What are we *not* modeling? So far, we have only included rigid body effects—what are other effects that might influence the behavior of the system. List three other effects you believe would most influence the system that we have *not* included in our mathematical description of the system.

Task: Write the model both in state-space form and as a transfer function for all θ_i outputs. you may wish (in fact, I highly recommend) using a symbolic software package such as *Mathematica* to compute the transfer function from the state space model.

B.2.2 Task #2–Simulation

Task: Implement your model both as a transfer function and as a state-space system in LabVIEW’s Simulation Module. You should initially use the parameters that you have guessed, but you should be prepared to change these parameters. (If you do not have a transfer function at this point, you should do the rest of the lab first using a state space model—you can create and use the transfer function outside of class if necessary.)

Task: Give the (open-loop!) model a step input. Does the model respond the way you expect it to? You may, if you wish, come up and experiment by hand with the torsional disk systems (while they are turned off).

Task: Start the simulation of your model at a nontrivial initial condition (in the state-space model) to see if that tells you anything else useful.

Task: How can other measurements help you, such as: natural frequency of oscillation, rate at which the system response decays due to dissipation, and sluggishness of response of the system to input torque? Can you think of anything else that would allow you to evaluate the correctness of your model?

Task: For each case, turn in a plot that includes the relevant *output* of the system and the input, explaining what the result implies for your choice of parameter.

Note that this part of the lab is very open-ended. It is intended to be open-ended. Don’t get frustrated!

Task: Now, based on your findings, choose different (or possibly the same, if you were very lucky) parameters of the system

Variable	Value
J_1	
J_2	
J_3	
c_1	
c_2	
c_3	
k_1	
k_2	

B.2.3 Task #3–Sensitivity to Feedback

One of the main things you’ll learn in this class is that feedback is a very powerful tool, with both good and bad consequences. In the case of the torsional disk system, a nominally stable system can be made unstable. Assume for the rest of the lab that you have a proportional

controller of constant K . Is the system, with the parameters you have chosen, stable? Consider the following questions:

Task: Write the equations of motion for just the bottom disk. (I.e., you are setting $J_2 = J_3 = 0$.) Using the parameter values you decided upon, design a controller K that is stable for disk 1 and has performance you find satisfactory.

Task: What happens when you use this controller on the original three disk system? Try to give a physically intuitive explanation of what you observe in the simulation.

Task: Can you change the *parametric* properties (i.e., the inertias, damping, and spring constants) of the torsional system such that you are satisfied with the result? E.g., do you get a stable, fast response with little overshoot? What do such changes mean physically for the system?

B.3 Lab #2 Final Tasks

Task: For θ_3 , simulate in LabVIEW the transfer function model of the torsional disk system. Provide a simulation of the system's response to a step input.

Task: Do your two models (the state-space model and the transfer function model) provide the same numerical results? Be precise! If they are different, why? If they are the same, how are you sure?

Task: Write a brief overview of what you think an effective procedure would be to estimate the parameters $J_1, J_2, J_3, c_1, c_2, c_3, k_1, k_2$.

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

Appendix C

Lab #3: System Identification of the Torsional Disk System

Objective

The purpose of this lab is to familiarize you with the functioning of the EPS Torsional Disk System (TDS) and learn how to perform a system identification on it. You have already done a somewhat heuristic version of this in the previous lab—this time you will systematically identify the parameters—in particular the parameters you cannot compute beforehand based on physical principles (e.g., the torsional spring constants, the inertia of the disk, and the dissipation constants). You will also need to establish the hardware gain for the system. Note that it will serve you well to be familiar with this process—the four ECP units are somewhat different from each other and will typically change their parametric values over time, so you may find that you need to re-identify these systems as the lab progresses. (Hopefully you will not need to, however.)

NOTE THAT THIS LAB WILL BE A TWO-WEEK LAB.

WARNING:

Please do not break the TDS systems. If you have questions, or are unsure of a next step, please let us know. However, the most important things are:

1. *never run the system while it is clamped down!* If you follow this one rule, it is extremely unlikely (though sadly not impossible) that you will break the system;
2. do not twist the disks more than 30 degrees relative to each other as you can damage the torsional rod this way as well.

C.1 Pre-Lab Tasks

These tasks must be completed and (the concrete portions) shown to either the TA or the instructor before you may use the hardware.

Task: Read this entire document.

Task: Create a VI that allows you to write data to a file. Create another one that allows you to read the data and plot it. Refer to Section 3.4 for details on how to do this.

Task: We will provide you with the VI infrastructure for talking to the hardware. Create a single pulse using signal generation tools from the simulation module.

Task: Model the bottom and top disk as second order systems assuming that the middle disk is clamped down. (You will clamp the middle disk while doing the system identification, so you will need this model.)

Task: Write down the impulse response of the bottom disk and the top disk. What does this correspond to physically? Can you simulate it? Can you experimentally produce the impulse response?

C.2 Tasks

C.2.1 Task #1—Using the Hardware

There are a number of tasks here that have to be done during the system identification process. The basic idea is that you will clamp the middle disk in order to separate the dynamics of the top disk from the bottom disk. You will then estimate the spring constants k_1 and k_2 by looking at the period of oscillation and you will estimate the dissipation constants by looking at the decay envelope of the system as oscillations die out. First, however, you need to be sure you can get data for the system.

1. Clamp the center disk and keep the hardware controller *turned off*.
2. Put four masses on both the top and bottom disks 9.0 cm away from the center of the disk. (The rings in the disk are 1.0 cm apart.)
3. Test that you can read all three encoder values for a 10 second time period.
4. When you start the experiment in LabVIEW, the encoder should automatically zero so that all the encoders are at zero. Let us know if they don't.
5. Plot the output of one of the encoders. Keep in mind that the encoder will give you 16,000 counts for every revolution. (You should convert to radians.)
6. Save both the time data and encoder data for one of the encoders in a file.
7. Read your results back into LabVIEW and plot them.

C.2.2 Task #2—Creating a System Identification Procedure

There are several parameters you need to calculate to have a complete model of the ECP unit. These are: the inertias J_1, J_2, J_3 of the disks, the dissipation coefficients c_1, c_2, c_3 , the spring constants k_1 and k_2 , and lastly the hardware gain k_{hw} that scales voltages from the FPGA to torque. For all that follows, assume that $c_2 = c_3$.

Task: The inertia of each disk is equal to the inertia due to the brass weights plus the inertia of the unloaded disk. That is, $J = J_{disk} + nJ_{brass}$ where n is the number of weights on a given disk. For now, assume that $J_{disk} = 0$. The brass weights have a mass (incl. bolt & nut) of $500g$ ($\pm 5g$) and a diameter of $5.00cm$ ($\pm 0.02cm$). Use this to calculate the inertia for the disks with the brass weights 9 cm out from the center of the disk.

Task: Using the impulse response you calculated for both the bottom and top disk, how can you calculate the associated σ and ω_d using experimental data?

Task: Given σ and ω_d , derive a formula for the dissipation constant c and the spring constant k for both the bottom disk and the top disk. This should now mean that you have a way to calculate J_1, J_2, J_3, c_1, c_3 , and k_1, k_2 . If you assume that $c_2 = c_3$ (which it roughly is), then you can calculate all these parameters from some data.

C.2.3 Task #3—System Identification Experiment

Task: Discuss your system identification approach with the TA or instructor before taking data.

Task: Clamp down the middle disk using the clamp provided.

Task: Take the data you decided upon and calculate $J_1, J_2, J_3, c_1, c_2, c_3$, and k_1, k_2 .

UNCLAMP THE ECP UNIT NOW!!!!!!

C.2.4 Task #4–System Identification Results

Write down the values of the parameters you computed here.

Variable	Value
J_1	
J_2	
J_3	
c_1	
c_2	
c_3	
k_1	
k_2	

This is probably a good potential stopping point for day one.

C.2.5 Task #5–Hardware Gain

Task: Decide on a method of calculating the hardware gain k_{hw} using a pulse. There are many ways of doing this—whatever suits you is fine. The voltage is divided into 3270 increments, so if you want X volts you will have to multiply it by the gain 3270 before sending the signal to the FPGA.

Task: Take the data you have decided is necessary and compute k_{hw} .

Task: Is the relationship between voltage and output torque really a linear relationship? (I.e., is $\frac{F}{V}$ really a constant?) Take several pieces of data to decide. If it is not linear, is it “close enough” to linear? Why/why not?

C.2.6 Task #6–System Identification Analysis

You now have a nominally identified system! The question is whether these parameters are enough to give you a good model of what the system *actually* does.

Task: Use your state-space implementation of the system to simulate the system with the correct parameters you have just computed.

Task: Design two experiments that you can use to verify your model against the experiment. One should involve no applied torque (just setting the initial conditions) and the other should involve an applied torque.

Task: We assumed that $J_{disk} = 0$ when we were calculating the inertia. Either design an experiment to calculate the actual value of J_{disk} or analytically justify why it is unnecessary.

Task: How close is your model to the real system? Describe what differences there are. Can they be fixed? If not, how will they affect the controller design?

This problem is open-ended. You need to justify to the best of your ability the fact that your model adequately represents the real system.

Task: Lastly, decide on a way to approximate the error in your estimates for the parameters in the system (J_i , c_i , k_i , k_{hw}).

C.3 Things You May Want To Know

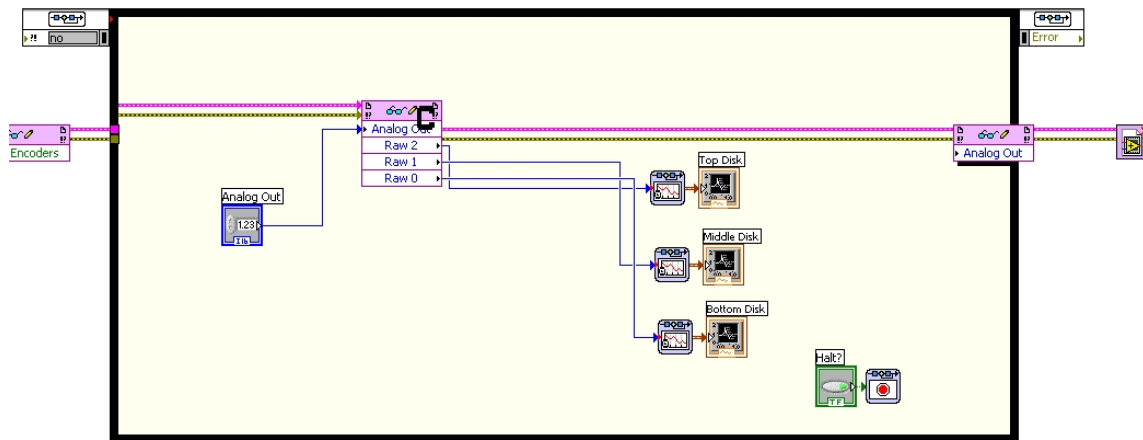


Figure C.1: The template you will be using

All of the FPGA code for this lab and most (possibly all) of the subsequent labs will be written for you and loaded onto the boards in advance.

Template vi's, in normal LabVIEW for windows, have been created for your use in this lab. To use them, go to **Desktop--Modules--ECEN Control Systems--Torsional Interface** and copy the files to a directory on your "Z"-drive. You may rename the file "torisonal_plant_template.vi" to whatever you wish, but do not rename the other files.

When the VI is first run, LabVIEW may report a "VISA: Code Module could not be loaded." In this case, close LabVIEW, open the "Measurement and Automation" on the desktop. Close the program once it is loaded and restart LabVIEW. This should fix the error.

Since the data acquisition and analog out (which will be how you send an input to the motor in "Hardware Gain on Torque") are within a Simulation Loop, you can use the same

vi's that you have been using to do your Simulations, i.e. step input, simulation time graph, etc.

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

Appendix D

Lab #4: Open-Loop and Hardware-Based PID Control Design

Objective

The purpose of this lab is to use EPS Torsional Disk System (TDS) to learn how to apply some simple control techniques to a reasonably high-order, non-trivial system. As such, there will be no pre-lab simulation, although there will be some simulation in your analysis of your results. We are also attempting to create a benchmark for how well one can do using “hack” techniques that involve no analysis.

The lab will consist of two main components. First, you will figure out what open-loop control you can apply to get the ECP unit to rotate 180 degrees and then rotate back 180 degrees with four weights on the bottom disk and two weights on the middle and top disks. Then, based on experiment, you will design P, PD, and PID controllers to meet some specifications and do the same experiment.

Please always run the ECP unit with someone holding the power button, in case something happens to go wrong.

NOTE THAT THIS LAB WILL BE A ONE-WEEK LAB.

Warning:

As before, please do not break the TDS systems. If you have questions, or are unsure of a next step, please let us know. However, the most important thing is to *never run the system while it is clamped down!* If you follow this one rule, it is extremely unlikely (though sadly not impossible) that you will break the system.

D.1 Pre-Lab Tasks

If you wish, you may want to create the open-loop VI and PID VI's ahead of time. Aside from that, there are no pre-lab tasks.

D.2 Tasks

D.2.1 Task #1—Open-Loop Control

Attach four weights to the bottom disk, two disks to the middle disk, and two disks to the top disk, all at 7cm from the center.

Task: You have the transfer function $H(s)$ going from the control input to the bottom disk output. Use this to construct an open loop control that moves the bottom disk from an initial orientation of 0 to a final orientation of π and back to 0 in 5 seconds with an error of 2 degrees ($= 2\pi/180 \text{ rad}$). Keep in mind that if you invert a transfer function you will have to add some filters to make it proper. Repeat this experiment 10 times. What is the average error of the outcome?

Task: Move the weights to 9cm and perform the experiment again using the same open loop control inputs. How much more error do you get?

D.2.2 Task #2—P, PD, PID Control

Before running these experiments, please let the TA or myself see your LabVIEW code. Also, be prepared for the fact that you *may destabilize the system*, so don't be caught unaware! This is because you are introducing feedback into a system, and feedback can make an otherwise stable system unstable.

Task: With the weights at their 7cm location, implement a proportional controller $C(s) = K$ for the bottom disk output (read the end of the lab to see how to do this). Try to get a rise time of 1s and less than 30% overshoot and settling time of 3s and $e_{ss} \leq 2\text{deg}$ with respect to a step input. Once you have your system stabilized using your controller, *gently* rotate the disk with the feedback engaged. It should resist your push in much the same way a spring does.

Task: With the weights still at their 7cm location, implement a PD controller ($C(s) = K + K_D s$) to meet the specifications.

Task: With the weights at their 7cm location, implement a PID controller ($C(s) = K + K_D s + K_I \frac{1}{s}$) to meet the specifications.

Task: Move the weights to 9cm . How does your performance change?

Task: Change the output of the system to be the top disk. *Be careful here* because you might destabilize the system. Does this make the control design easier or harder? How does your controller for the bottom disk perform for the top disk? What are the qualitative differences?

How do you explain the differences? What values of K , K_D and K_I in $C(s) = K + K_D s + K_I \frac{1}{s}$ stabilize the system?

D.2.3 Task #3—Simulation

Task: Simulate your controller designs with the parameters you found in Lab 3 and with the simulations you built in Lab 2. Compare the output of the actual physical system to the simulation. How should one compare two different responses? Relative to such a comparison, how close is the simulation to the real system? What parameters need to be fixed?

Task: In simulation, can you improve your PID design (for either output)?

Task: At some point in this lab, you have almost certainly destabilized the system by making some parameter too large. Use the *rlocus* command in MathScript (or the Interactive Root Locus Tool in LabVIEW) to find the root locus for the system relative to this parameter. Can you explain the instability you saw using the root locus?

D.3 Things You May Want To Know

Here are some LabVIEW hints for this lab.

1. Use a Create Control command for the reference input—this will allow you to play the with controller “on the fly.”
2. Be very careful to make sure all your units are the same.

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

Appendix E

Lab #5: Root Locus Design

Objective

The purpose of this lab is to use the EPS Torsional Disk System (TDS) to learn how to apply root locus techniques to a real, physical system. You have already seen how PID works on the system in the last lab. Now, you are going to see some of your experience validated analytically by analyzing the root locus for the ECP system.

The lab will consist of three main components. First, you will determine the hardware gain that you think is appropriate for the system and what error bounds are appropriate for the parameters we are using. Second, you will determine the root locus for the system given the “collocated” outputs (the outputs that are located at the same place as the input) and the “noncollocated” outputs (the outputs that are located somewhere other than the input location). (This will help you see why stabilizing the top disk is intrinsically harder than stabilizing the bottom disk.) Lastly, you will iteratively determine what “good” values of a PID controller are based on the root locus.

NOTE THAT THIS LAB WILL BE A TWO-WEEK LAB.

Warning:

As usual, please always run the ECP unit with someone holding the power button, in case something happens to go wrong.

E.1 Pre-Lab Tasks

Read this **entire** document before starting the lab. Aside from that, you may wish to create a VI that will take the model of your system and plot a root locus (see Section 4.1.1). I strongly recommend that you use your state-space representation for this purpose. However, you may use either the state-space or transfer function representations, as you see fit.

E.2 Tasks

E.2.1 Task #1—Making Sure Your Simulation Works

The purpose of this part of the lab is to make sure that your simulation is reasonably related to the real physical system.

Task: Using the hardware template, make sure that you are running the experimental system with a time steps of 0.01 s.

Task: Run the experiment in one VI and the simulation in another VI. I would recommend a closed-loop step response of roughly $\frac{\pi}{2}$ rad and an open-loop pulse of 0.3 Nm for 0.5 s would probably be a good combination. Make sure that your simulation and the experiment have roughly the same behavior. If necessary adjust the hardware gain block to make your simulation match the experimental behavior as closely as possible.

E.2.2 Task #2—Root Loci for Disk 1 and Disk 3 Outputs

Task: Draw a block diagram that represents the system with a proportional controller. Label all the blocks and all the lines that connect the blocks. Lastly, give the physical units for each line.

Task: In LabVIEW, create the root locus for each output (disk 1 and 3). Which system is “more” stable using proportional feedback? Defend your answer. Test this hypothesis experimentally. For each disk, experimentally estimate the value of K where the system destabilizes. You may approach this estimation problem in whatever way you find most convenient—just detail it in your writeup.

Task: Now assume that you have a PD controller of the form $C(s) = K(1 + K_D s)$ so that K_D represents that *ratio* of the damping to the gain. Assuming that K_D varies from 0 to 1, plot the root locus for five relevant ratios that you choose. (Again, do this for both disk 1 and disk 3 outputs.) What is the best choice from your perspective? Defend your rational of your choice and test it experimentally. Explain your experimental results.

Task: Based on the root locus of your choice of ratio, try to estimate what value of K and K_D would give a rise time of 1 second and less than 30% overshoot. (Yes, again for both disk 1 and disk 3 as outputs.) To do this, you may wish to use the “Interactive Root Locus”

that allows you to see how the poles change as the gain K varies. Implement your design in hardware. How close do you get?

Task: Based on the root locus for each output and your choice of K and K_D , can you add an integral term K_I to your PID controller in order to get rid of steady-state error?

E.2.3 Task #3–Lead/Lag Controller

Task: We know that PID controllers can be approximated by “Lead/Lag” controllers. How does your root locus analysis change if you replace the PID controller with a Lead/Lag controller?

E.2.4 Task #4–Root Loci for Uncertainties

Task: For your choice of K and K_D above, plot the root locus for two parameters you think introduce the largest amount of uncertainty. For instance, you can plot the root locus with respect to c_1 and with respect to k_1 , if you think these add the most uncertainty. You may find *Mathematica* or another symbolic toolbox helpful for this problem.

Task: Based on the root locus, do these parameters represent a threat to the stability of the system (for either disk 1 or disk 3 as outputs)?

Task: Based on the root locus, how do these parameters affect the performance of the system?

E.2.5 Task #5–Design for and Difficulties with Disk 3

Task: Can you design a different compensator (perhaps with imaginary poles) that makes the system stable with better performance? Can you improve upon your PID controller performance? Try this first using the root locus tool, then in simulation, and finally (if you are convinced you have a good design) try it in hardware. If you didn’t get a design you liked, discuss what the obstacles were and what you tried to use to overcome them.

E.3 Useful Things to Know

1. The analysis section of the LabVIEW manual on Moodle has been updated to show you how to use LabVIEW’s root locus tools.
2. We highly recommend using the new “Torsional Plant Interface Template” to access the FPGA. The new template provides more information, better error checking, and doesn’t need the Local Variable Feedback hack used in Lab 4.

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

Appendix F

Lab #6: Frequency Domain Compensator Design

Objective

The purpose of this lab is to expose you to frequency domain design techniques. In particular, you will use Bode plots and Nyquist plots to determine the stability and robustness of a design. Because you are now all largely familiar with LabVIEW, the description of this lab will be kept rather brief. However, keep in mind that this is nevertheless a reasonably long lab—don't put off the work until the second week!

NOTE THAT THIS LAB WILL BE A TWO-WEEK LAB.

Warning:

As usual, please always run the ECP unit with someone holding the power button, in case something happens to go wrong.

F.1 Pre-Lab Tasks

Read this entire document before starting the lab. A significant part of this lab can be completed without hardware, so I recommend you complete as much of it as you can before coming to lab.

F.2 Tasks

F.2.1 Task #1—Lead Controller for Disk 1

Task: Design a Lead controller for Disk 1 as the output and plot its root locus and its Bode plot.

Task: What is the associated gain margin (GM) and phase margin (PM)?

Task: How could you change your Lead controller to improve your margins? Can you do so while simultaneously increasing your overall system gain?

F.2.2 Task #2—Controller Design for Disk 3

Task: What is difficult about using Disk 3 as the output instead of Disk 1? What happens in the root locus if you try and use a Lead compensator in this case?

Task: Plot the Bode plot for the open loop system with the third disk as the output. Is it stable? How much gain K will destabilize it, if K comes from a proportional feedback controller with unitary feedback?

Task: Plot the Nyquist plot for the open loop system. What does this tell you about the system stability?

Task: Print the Nyquist plot and Bode plot and draw the gain margin and the phase margin into both plots for a particular value of K (a value close to what you used when you first stabilized this system with a proportional controller).

Task: Design a higher performance controller that allows you to push the overall system gain up higher while not destabilizing. You may find the notion of a *notch filter* to be useful here. (This is described in FPE, and the entire description there you may find useful.) Can you improve upon your PID controller performance? In particular, make sure that you maintain your gain and phase margin at the “reasonable level” you just determined. Test your design in simulation. What do you think are limits on the performance that this system can achieve?

Task: At this stage, the SISO tool in MATLAB (command: *sisotool*) will probably be very handy—you will probably want to take MathScript code and copy it to MATLAB. Use the SISO Tool in MATLAB to improve your controller design from the previous section. When moving poles and zeros around, *only* move the poles and zeros of your controller, not the plant itself!

Task: What is the gain margin and phase margin for your final controller design?

Task: Test your design in experiment. If there are differences between the experiment and the simulation, explain them.

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

Appendix G

Lab #7: State Space Compensator Design

Objective

The purpose of this lab is to expose you to state space design techniques. In particular, you will use LQR techniques to design controllers and estimators. Because you are now all largely familiar with LabVIEW, the description of this lab will be kept rather brief. However, keep in mind that this is nevertheless a reasonably long lab—don't put off the work until the second week!

NOTE THAT THIS LAB WILL BE THE LAST LAB.

Warning:

As usual, please always run the ECP unit with someone holding the power button, in case something happens to go wrong.

G.1 Pre-Lab Tasks

Read this entire document before starting the lab.

G.2 Tasks

1. The code that you will see will be found in *students-simulation.vi*, *students-hardware.vi*, and *students-estimator.vi*. You will only need to edit the mathscript in *students-simulation.vi* and *students-estimator.vi*.
2. Note that you will *only* change things in MathScript in this lab. Please do not change any of the graphical LabVIEW code except for rewiring the diagram to change full state feedback to estimated state feedback (as seen in Fig.G.2).
3. Lastly, you will be given values for $c_{1,2,3}$ and $k_{1,2}$ for each torsional disk system. Also, the hardware gain is now calculated within the code—you don't need to address it.

G.2.1 Task #1—State Space simulation

Task: Copy the folder `state-space` from Desktop/ITLL Modules/ECEN Control Systems to your Z: drive.

Task: In *students-simulation.vi*, replace the second-order model with your model of the torsional disk system. In *students-estimator.vi*, change the state feedback gain K (near “[MARKER 1]”) to be the appropriate length vector (e.g., it should match the dimension of the state model) made up of zeros. On the last line (near “[MARKER 3]”), set “Nbar=1”—this will allow you to track a reference input. (This hack will allow your code to compile, though there will be some errors due to the LQR control and estimation calculations not matching the dimension of K . You will learn by next week what Nbar is and how to compute the controller, estimator, and Nbar without errors.)

Task: Simulate some open loop response (using the top disk as the output) with nontrivial initial condition to verify that your model is functioning properly.

G.2.2 Task #2—Controller Design

Now that you have a model working, you should not need to change the *student-simulation.vi* file any more. Instead, everything else should occur within the *student-estimator.vi* file, where both the controller and estimator are designed. First you will design a state-space controller for stabilizing the top disk, then you will design a state-space estimator. For the controller, you should assume you have access to full state feedback—*make sure the block diagram reflects this*.

Task: Implement a proportional controller and a PD controller in state space form. (I.e., You will have to replace your K of zeros with your own definition of K that implements a PD control law.) Simulate the closed loop response. Is it sensitive to changes in system parameters? Disable the reference input and study your controller’s response to initial conditions only by setting “Nbar=0” in *students-estimator.vi*.

THIS IS PROBABLY A GOOD STOPPING POINT FOR THE FIRST WEEK. HOWEVER, YOU SHOULD HAVE DONE MOST OF THE SIMULATION COMPONENTS OF THIS LAB BY THE NEXT TIME YOU MEET.

Task: Design a controller using LQR (“lqr()” in MATLAB/MathScript)) that keeps the total control effort for a $1rad$ step response less than $2Nm$ with a rise time of $t_r < 0.75s$, overshoot $M_p < 25\%$, $e_{ss} < 5\%$, and settle time $t_s < 2s$. To track a reference, you will need to comment out the “[MARKER 3]” line so that Nbar is not disabled.

G.2.3 Task #3—Estimator Design

Task: Rewire the simulation so that it is using estimated state for purposes of control calculation, not the actual state.

Task: Using the top disk as the output, design a state estimator by placing the poles at desired locations using the command “place()”.

Task: Simulate the system using your controller and the estimated state. Is your performance as good? (You will have to comment out the line at “[MARKER 2]”.)

Task: Using the formula for the controller $C(s)$ given controller K and estimator L , calculate the phase margin and gain margin of your system?

Task: Does your controller still work when you vary parameters somewhat? If it does, you need to iterate!

G.2.4 Task #4—Hardware Test

Task: If you are confident that your controller is working properly in simulation, verify it in hardware. This can be done using *students-hardware.vi*. Set the hardware gain for your plant in the VI’s wiring diagram and run it—it will reference the *students-estimator.vi* you have already created.

Task: What happens if you change the sample rate from $200Hz$ to $500Hz$? to $1000Hz$? Why?

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.

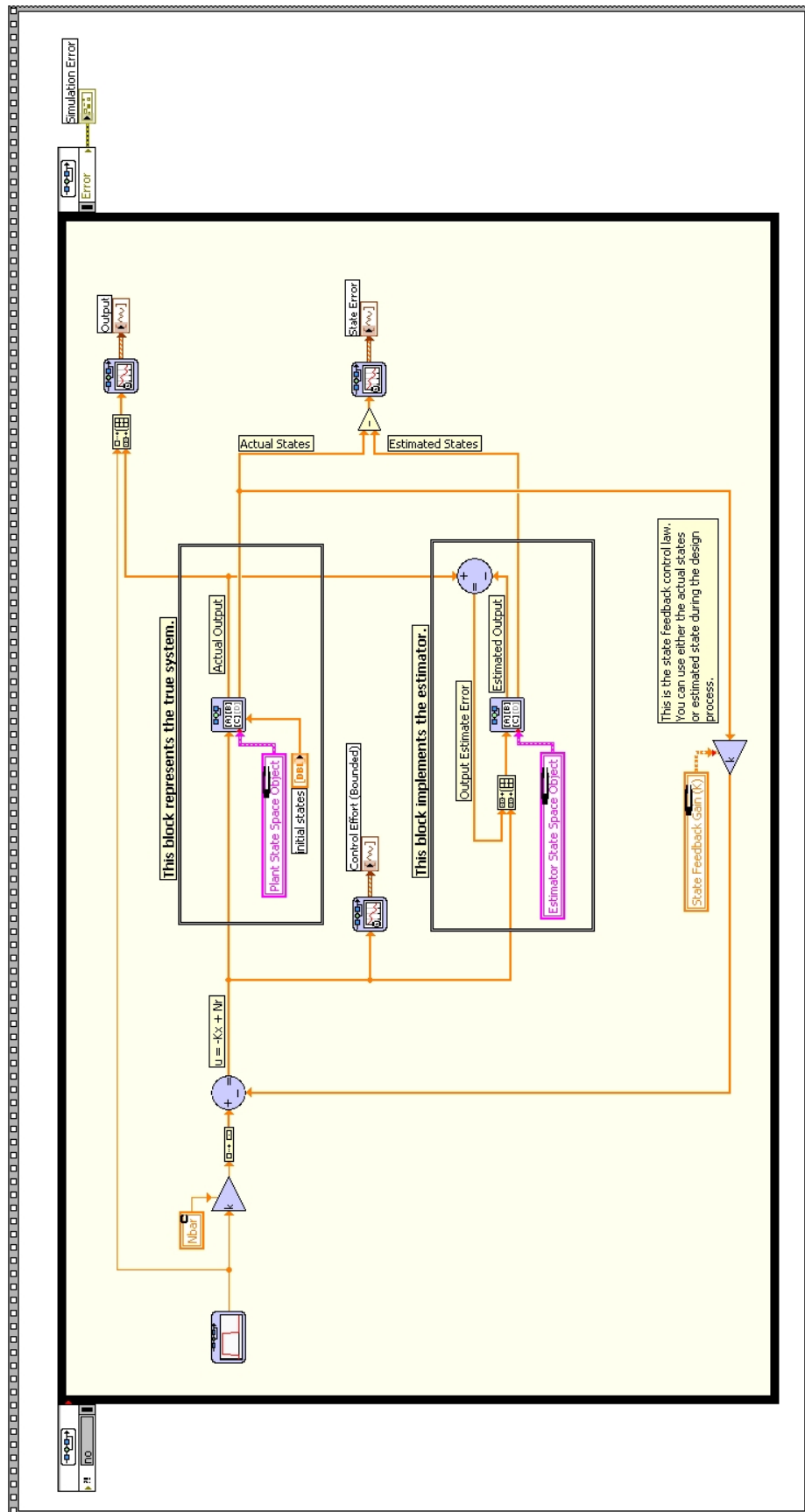


Figure G.2: State Space Simulation

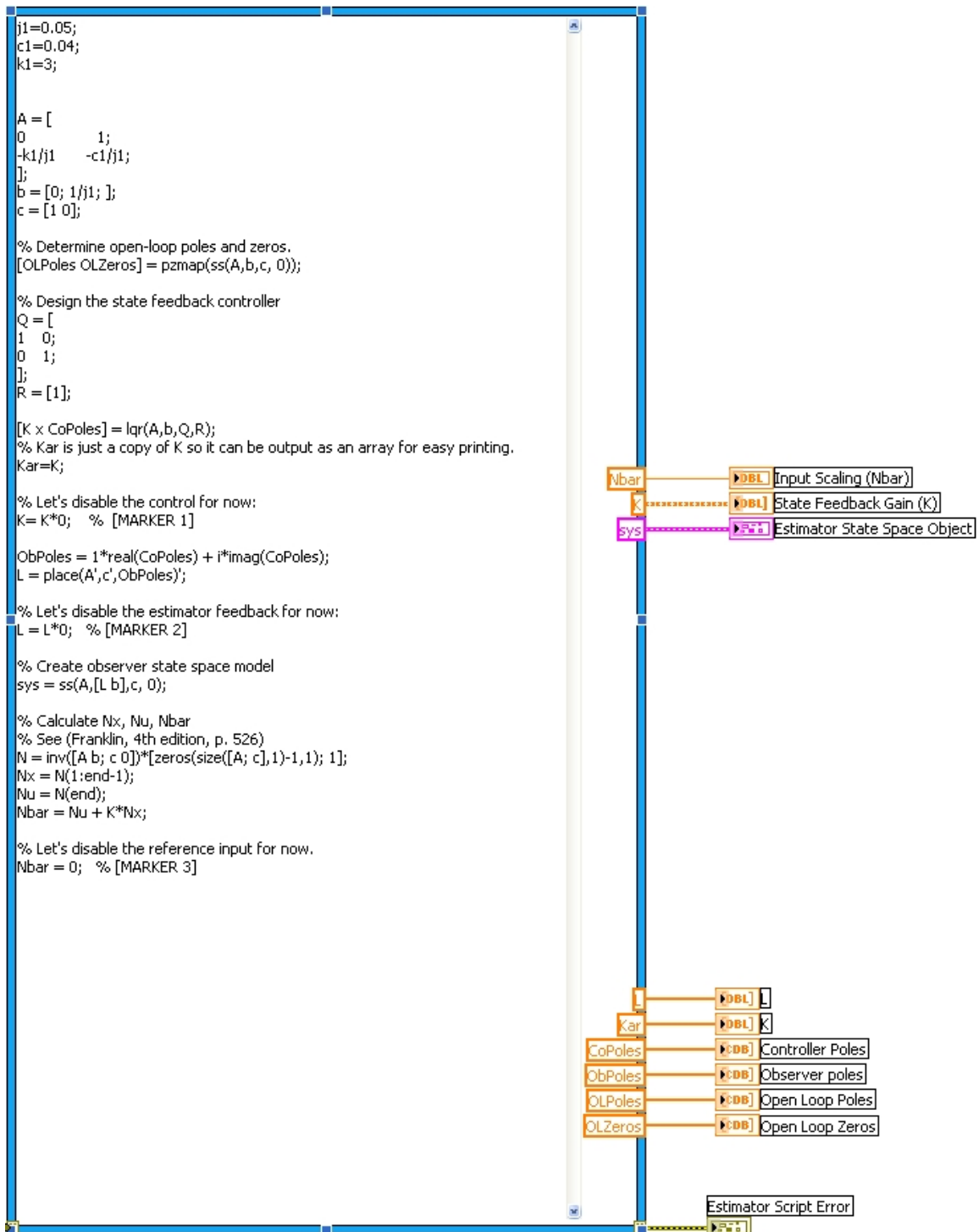


Figure G.3: State Space Estimation and Control